MANUDAX — NEDERLAND B.V.
Meerstraat 7, PB 25, 5473 ZG Heeswijk (N.B.) - Holland - Tel. 04139-1252* Telex 50175

MC68000

Advance Information

16-BIT MICROPROCESSING UNIT

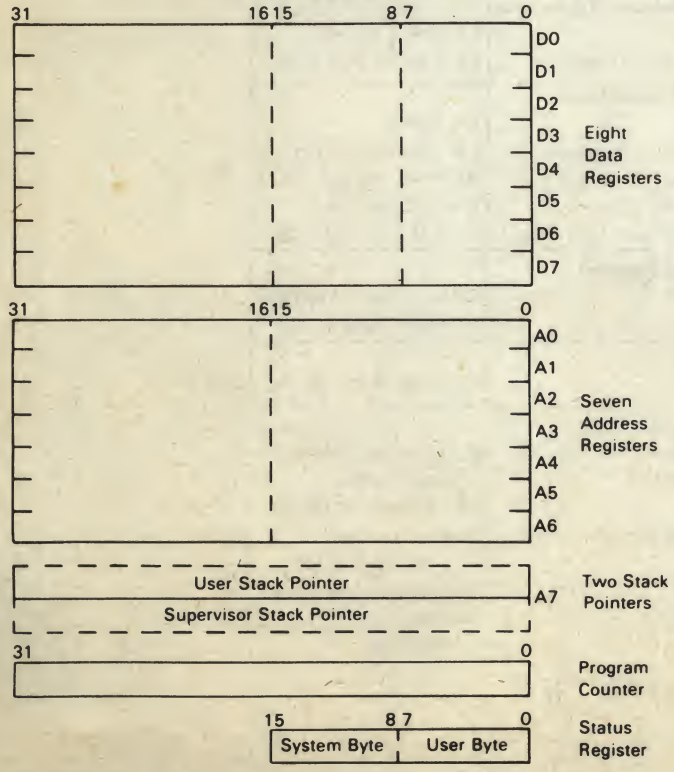
Advances in semiconductor technology have provided the capability to place on a single silicon chip a microprocessor at least an order of magnitude higher in performance and circuit complexity than has been previously available. The MC68000 is the first of a family of such VLSI microprocessors from Motorola. It combines state-of-the-art technology and advanced circuit design techniques with computer sciences to achieve an architecturally advanced 16-bit microprocessor.

The resources available to the MC68000 user consist of the following:

- 32-Bit Data and Address Registers
- 16 Megabyte Direct Addressing Range
- 56 Powerful Instruction Types
- Operations on Five Main Data Types
- Memory Mapped I/O
- 14 Addressing Modes

As shown in the programming model, the MC68000 offers seventeen 32-bit registers in addition to the 32-bit program counter and a 16-bit status register. The first eight registers (D0-D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) data operations. The second set of seven registers (A0-A6) and the system stack pointer may be used as software stack pointers and base address registers. In addition, these registers may be used for word and long word address operations. All 17 registers may be used as index registers.

PROGRAMMING MODEL



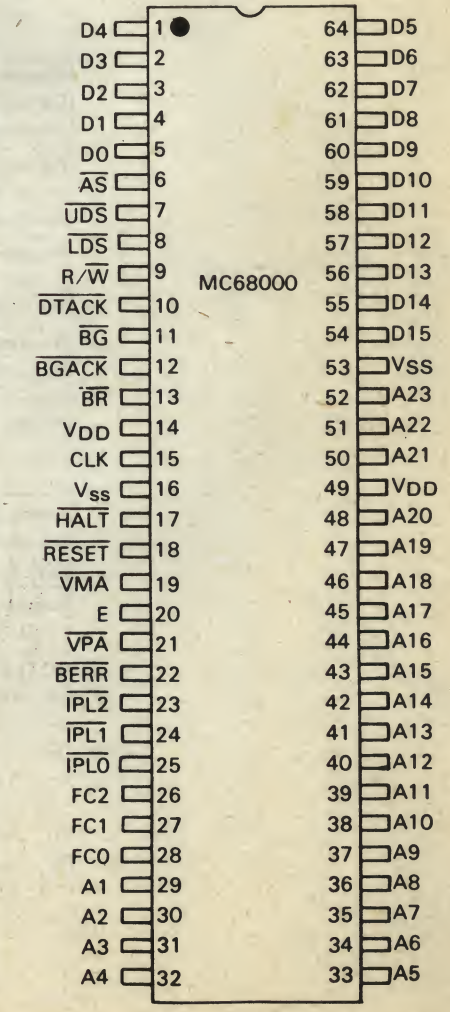
HMOS

(High Density N-Channel, Silicon-Gate Depletion Load)

16-BIT MICROPROCESSOR

L SUFFIX
CERAMIC PACKAGE
CASE 746-1

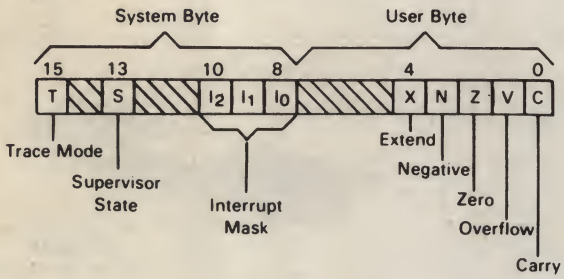
PIN ASSIGNMENT



A 23-bit address bus provides a memory addressing range of greater than 16 megabytes. This large range of addressing capability, coupled with a memory management unit, allows large, modular programs to be developed and operated without resorting to cumbersome and time consuming software bookkeeping and paging techniques.

The status register contains the interrupt mask (eight levels available) as well as the condition codes; extend (X), negative (N), zero (Z), overflow (V), and carry (C). Additional status bits indicate that the processor is in a trace (T) mode and/or in a supervisor (S) state.

STATUS REGISTER



Five basic data types are supported. These data types are:

- Bits
- BCD Digits (4-bits)
- Bytes (8-bits)
- Word (16-bits)
- Long Words (32-bits)

In addition, operations on other data types such as memory addresses, status word data, etc. are provided for in the instruction set.

The 14 addressing modes, shown in Table 1, include six basic types:

- Register Direct
- Register Indirect
- Absolute
- Immediate
- Program Counter Relative
- Implied

Included in the register indirect addressing modes is the capability to do postincrementing, predecrementing, offsetting and indexing. Program counter relative mode can also be modified via indexing and offsetting.

TABLE 1 — DATA ADDRESSING MODES

Mode	Generation
Register Direct Addressing	
Data Register Direct	EA = Dn
Address Register Direct	EA = An
Absolute Data Addressing	
Absolute Short	EA = (Next Word)
Absolute Long	EA = (Next Two Words)
Program Counter Relative Addressing	
Relative with Offset	EA = (PC) + d_{16}
Relative with Index and Offset	EA = (PC) + (Xn) + d_8
Register Indirect Addressing	
Register Indirect	EA = (An)
Postincrement Register Indirect	EA = (An), An ← An + N
Predecrement Register Indirect	An ← An – N, EA = (An)
Register Indirect With Offset	EA = (An) + d_{16}
Indexed Register Indirect With Offset	EA = (An) + (Xn) + d_8
Immediate Data Addressing	
Immediate	DATA = Next Words(s)
Quick Immediate	Inherent Data
Implied Addressing	
Implied Register	EA = SR, USP, SP, PC

NOTES:

EA = Effective Address
An = Address Register
Dn = Data Register
Xn = Address or Data Register used as Index Register
SR = Status Register
PC = Program Counter
( ) = Contents of
d_8 = Eight-bit Offset (displacement)
d_{16} = Sixteen-bit Offset (displacement)
N = 1 for Byte, 2 for Words and 4 for Long Words
← = Replaces

The MC68000 instruction set is shown in Table 2. Some additional instructions are variations, or subsets, of these and they appear in Table 3. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned multiply and divide, "quick" arithmetic operations, BCD arithmetic and expanded operations (through traps).

TABLE 2 — INSTRUCTION SET

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
ABCD	Add Decimal with Extend	EOR	Exclusive Or	PEA	Push Effective Address
ADD	Add	EXG	Exchange Registers	RESET	Reset External Devices
AND	Logical And	EXT	Sign Extend	ROL	Rotate Left without Extend
ASL	Arithmetic Shift Left	JMP	Jump	ROR	Rotate Right without Extend
ASR	Arithmetic Shift Right	JSR	Jump to Subroutine	ROXL	Rotate Left with Extend
Bcc	Branch Conditionally	LEA	Load Effective Address	ROXR	Rotate Right with Extend
BCHG	Bit Test and Change	LINK	Link Stack	RTE	Return from Exception
BCLR	Bit Test and Clear	LSL	Logical Shift Left	RTR	Return and Restore
BRA	Branch Always	LSR	Logical Shift Right	RTS	Return from Subroutine
BSET	Bit Test and Set	MOVE	Move	SBCD	Subtract Decimal with Extend
BSR	Branch to Subroutine	MOVEM	Move Multiple Registers	Scc	Set Conditional
BTST	Bit Test	MOVEP	Move Peripheral Data	STOP	Stop
CHK	Check Register Against Bounds	MULS	Signed Multiply	SUB	Subtract
CLR	Clear Operand	MULU	Unsigned Multiply	SWAP	Swap Data Register Halves
CMP	Compare	NBCD	Negate Decimal with Extend	TAS	Test and Set Operand
DBcc	Test Cond., Decrement and Branch	NEG	Negate	TRAP	Trap
DIVS	Signed Divide	NOP	No Operation	TRAPV	Trap on Overflow
DIVU	Unsigned Divide	NOT	One's Complement	TST	Test
		OR	Logical Or	UNLK	Unlink

TABLE 3 — VARIATIONS OF INSTRUCTION TYPES

Instruction Type	Variation	Description	Instruction Type	Variation	Description
ADD	**ADD**	Add	**MOVE**	**MOVE**	Move
	ADDA	Add Address		MOVEA	Move Address
	ADDQ	Add Quick		MOVEQ	Move Quick
	ADDI	Add Immediate		MOVE from SR	Move from Status Register
	ADDX	Add with Extend		MOVE to SR	Move to Status Register
				MOVE to CCR	Move to Condition Codes
AND	**AND**	Logical And		MOVE USP	Move User Stack Pointer
	ANDI	And Immediate	**NEG**	**NEG**	Negate
				NEGX	Negate with Extend
CMP	**CMP**	Compare	**OR**	**OR**	Logical Or
	CMPA	Compare Address		ORI	Or Immediate
	CMPM	Compare Memory	**SUB**	**SUB**	Subtract
	CMPI	Compare Immediate		SUBA	Subtract Address
				SUBI	Subtract Immediate
EOR	**EOR**	Exclusive Or		SUBQ	Subtract Quick
	EORI	Exclusive Or Immediate		SUBX	Subtract with Extend

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{DD}	-0.3 to + 7.0	Vdc
Input Voltage	V_{in}	-0.3 to +7.0	Vdc
Operating Temperature Range	T_A	0 to 70	°C
Storage Temperature	T_{stg}	-55 to 150	°C

ELECTRICAL CHARACTERISTICS (V_{DD} = 5.0 Vdc ±5%; V_{SS} = 0 Vdc; T_A 25°C)

Characteristic		Symbol	Min	Typ	Max	Unit
Input High Voltage		V_{IH}	—	2.0	V_{DD}	Vdc
Input Low Voltage		V_{IL}	V_{SS} - 0.3	0.8	—	Vdc
Input Leakage Current	$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{IPL0}$-$\overline{IPL2}$, $\overline{VPA}$	I_{in}	—	1.0	—	μAdc
	$\overline{HALT}$, $\overline{RESET}$		—	2.0	—	
Three-State (Off State) Input Current	$\overline{AS}$, A1-A23, D0-D15 FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	I_{TSI}	—	2.0	—	μAdc
Output High Voltage (I_{OH} = -400 μAdc)	$\overline{AS}$, A1-A23, $\overline{BG}$, D0-D15, E, FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	V_{OH}	—	2.4	—	Vdc
Output Low Voltage		V_{OL}				Vdc
(I_{OL} = 1.6 mA)	$\overline{HALT}$		—	V_{SS} + 0.4	—	
(I_{OL} = 3.2 mA)	A1-A23, $\overline{BG}$, E, FC0-FC2		—	V_{SS} + 0.4	—	
(I_{OL} = 5.0 mA)	$\overline{RESET}$		—	V_{SS} + 0.4	—	
(I_{OL} = 5.3 mA)	$\overline{AS}$, D0-D15, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$		—	V_{SS} + 0.4	—	
Power Dissipation (Clock Frequency = 8 MHz)		P_D	—	1.2	—	W
Capacitance (Package Type Dependent) (V_{in} = 0 Vdc; T_A = 25°C; Frequency = 1 MHz)		C_{in}	—	10.0		pF

FIGURE 1 — RESET TEST LOAD

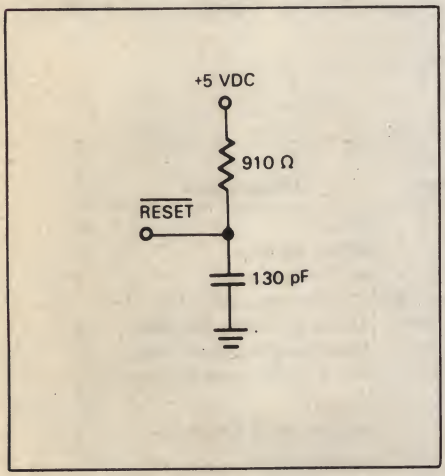


FIGURE 2 — HALT TEST LOAD

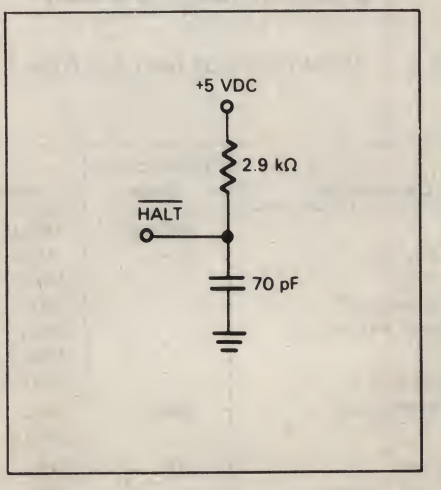
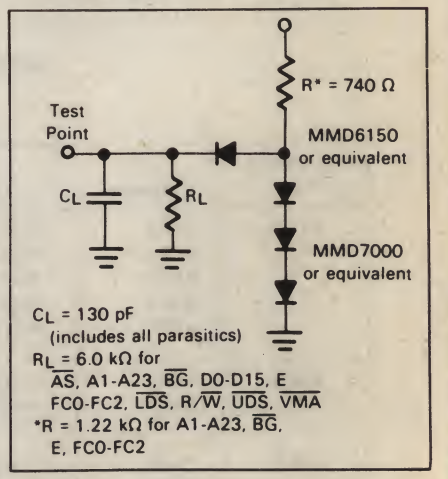


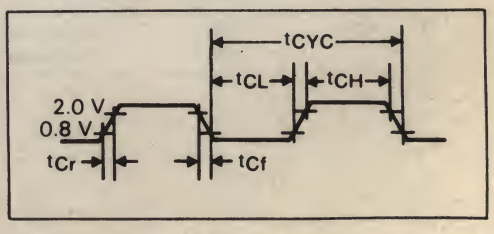
FIGURE 3 — TEST LOADS



CLOCK TIMING

Characteristic	Symbol	Min	Typ	Max	Unit
Frequency of Operation	F	—	—	8.0	MHz
Cycle Time	tCYC	—	125	—	ns
Clock Pulse Width	tCL	—	55	—	ns
	tCH	—	55	—	ns
Rise and Fall Times	tCr	—	—	10	ns
	tCf	—	—	10	ns

FIGURE 4 — INPUT CLOCK WAVEFORM



DATA ORGANIZATION AND ADDRESSING CAPABILITIES

The following paragraphs describe the data organization and addressing capabilities of the MC68000.

OPERAND SIZE

Operand sizes are defined as follows: a byte equals 8 bits, a word equals 16 bits, and a long word equals 32 bits. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. All explicit instructions support byte, word or long operands. Implicit instructions support some subset of all three sizes.

DATA ORGANIZATION IN REGISTERS

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers together with the active stack pointer support address operands of 32 bits.

DATA REGISTERS. Each data register is 32 bits wide. Byte operands occupy the low order 8 bits, word operands the low order 16 bits, and long operands the entire 32 bits. The least significant bit is addressed as bit zero; the most significant bit is addressed as bit 31.

When a data register is used as either a source or destination operand, only the appropriate low-order portion is changed; the remaining high-order portion is neither used nor changed.

ADDRESS REGISTERS. Each address register and the stack pointer is 32 bits wide and holds a full 32 bit address. Address registers do not support byte sized operands. Therefore, when an address register is used as a source operand, either the low order word or the entire long operand is used depending upon the operation size. When an address register is used as the destination operand, the entire register is affected regardless of the operation size. If the operation size is word, any other operands are sign extended to 32 bits before the operation is performed.

DATA ORGANIZATION IN MEMORY.

Bytes are individually addressable with the high order byte having an even address the same as the word, as shown in Figure 5. The low order byte has an odd address that is one count higher than the word address. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long word datum is located at address n (n even), then the second word of that datum is located at address n + 2.

The data types supported by the MC68000 are: bit data, integer data of 8, 16, or 32 bits, 32-bit addresses and binary coded decimal data. Each of these data types is put in memory, as shown in Figure 6.

FIGURE 5 — WORD ORGANIZATION IN MEMORY

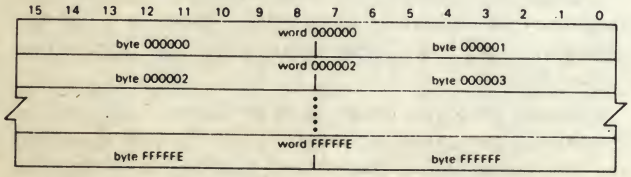
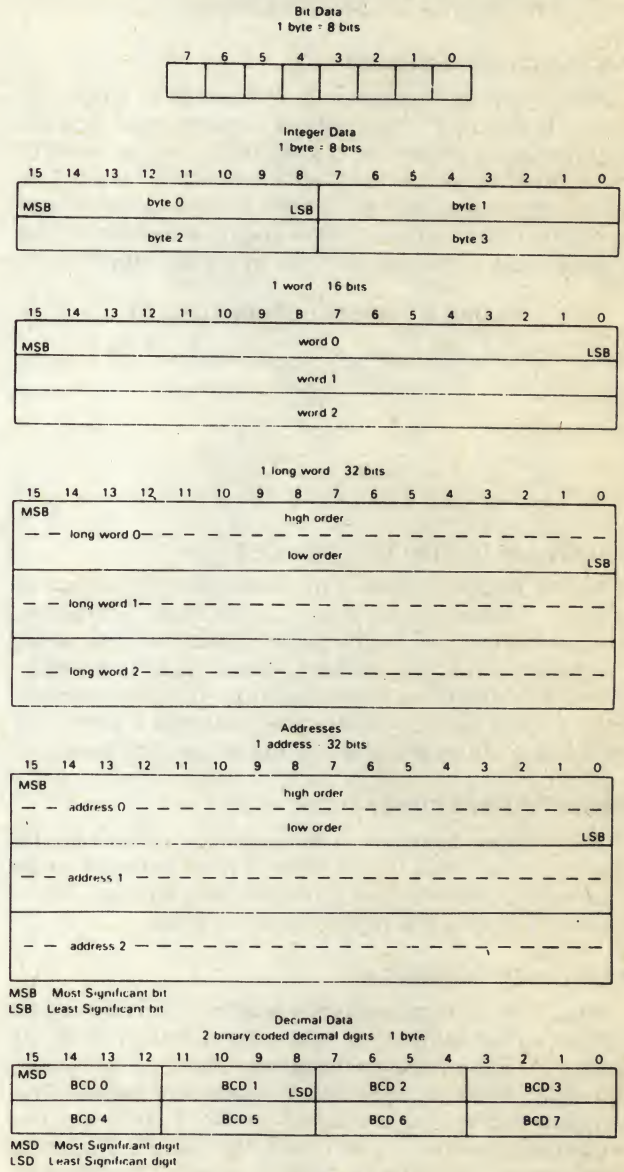


FIGURE 6 — DATA ORGANIZATION IN MEMORY



ADDRESSING

Instructions for the MC68000 contain two kinds of information: the type of function to be performed, and the location of the operand(s) on which to perform that function. The methods used to locate (address) the operand(s) are explained in the following paragraphs.

Instructions specify an operand location in one of three ways:

Register Specification — the number of the register is given in the register field of the instruction.

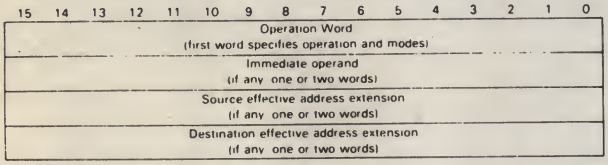
Effective Address — use of the different effective address modes.

Implicit Reference — the definition of certain instructions implies the use of specific registers.

INSTRUCTION FORMAT

Instructions are from one to five words in length, as shown in Figure 7. The length of the instruction and the operation to be performed is specified by the first word of the instruction which is called the operation word. The remaining words further specify the operands. These words are either immediate operands or extensions to the effective address mode specified in the operation word.

FIGURE 7 — INSTRUCTION FORMAT



PROGRAM/DATA REFERENCES

The MC68000 separates memory references into two classes: program references, and data references. Program references, as the name implies, are references to that section of memory that contains the program being executed. Data references refer to that section of memory that contains data. Generally, operand reads are from the data space. All operand writes are to the data space.

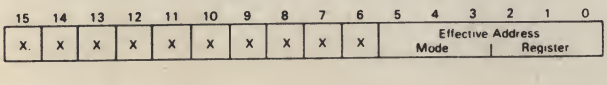
REGISTER SPECIFICATION

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 8 shows the general format of the single effective address instruction operation word. The effective address is composed of two 3-bit fields: the mode field, and the register field. The value in the mode field selects the different address modes. The register field contains the number of a register.

FIGURE 8 — SINGLE-EFFECTIVE-ADDRESS INSTRUCTION OPERATION WORD GENERAL FORMAT



The effective address field may require additional information to fully specify the operand. This additional information, called the effective address extension, is contained in the following word or words and is considered part of the instruction, as shown in Figure 7. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

REGISTER DIRECT MODES. These effective addressing modes specify that the operand is in one of the 16 multifunction registers.

Data Register Direct. The operand is in the data register specified by the effective address register field.

Address Register Direct. The operand is in the address register specified by the effective address register field.

MEMORY ADDRESS MODES. These effective addressing modes specify that the operand is in memory and provide the specific address of the operand.

Address Register Indirect. The address of the operand is in the address register specified by the register field. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

Address Register Indirect With Postincrement. The address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two, or four depending upon whether the size of the operand is byte, word, or long. If the address register is the stack pointer and the operand size is byte, the address is incremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

Address Register Indirect With Predecrement. The address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four depending upon whether the operand size is byte, word, or long. If the address register is the stack pointer and the operand size is byte, the address is decremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

Address Register Indirect With Displacement. This address mode requires one word of extension. The address of the operand is the sum of the address in the address register and the sign-extended 16-bit displacement integer in the extension word. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

Address Register Indirect With Index. This address mode requires one word of extension. The address of the operand is the sum of the address in the address register, the sign-extended displacement integer in the low order eight bits of the extension word, and the contents of the index register. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

SPECIAL ADDRESS MODES. The special address

modes use the effective address register field to specify the special addressing mode instead of a register number.

Absolute Short Address. This address mode requires one word of extension. The address of the operand is in the extension word. The 16-bit address is sign extended before it is used. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

Absolute Long Address. This address requires two words of extension. The address of the operand is developed by the concatenation of the extension words. The high-order part of the address is the first extension word; the low-order part of the address is the second extension word. The reference is classified as a data reference with the exception of the jump and jump to subroutine instrutions.

Program Counter With Displacement. This address mode requires one word of extension. The address of the operand is the sum of the address in the program counter and the sign-extended 16-bit displacement integer in the extension word. The value in the program counter is the address of the extension word. The reference is classified as a program reference.

Program Counter With Index. This address mode requires one word of extension. The address is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The value in the program counter is the address of the extension word. This reference is classified as a program reference.

Immediate Data. This address mode requires either one or two words of extension depending on the size of the operation.

Byte operation — operand is low order byte of extension word

Word operation — operand is extension word

Long operation — operand is in the two extension words, high-order 16 bits are in the first extension word, low-order 16 bits are in the second extension word.

Condition Codes Or Status Register. A selected set of instructions may reference the status register by means of the effective address field. These are:

ANDI to CCR
ANDI to SR
EORI to CCR
EORI to SR
ORI to CCR
ORI to SR

EFFECTIVE ADDRESS ENCODING SUMMARY. Table 4 is a summary of the effective addressing modes discussed in the previous paragraphs.

IMPLICIT REFERENCE

Some instructions make implicit reference to the program counter (PC), the system stack pointer (SP), the supervisor stack pointer (SSP), the user stack pointer (USP), or the status register (SR). Table 5 provides a list of these instructions and the registers implied.

TABLE 4 — EFFECTIVE ADDRESS ENCODING SUMMARY

Addressing Mode	Mode	Register
Data Register Direct	000	register number
Address Register Direct	001	register number
Address Register Indirect	010	register number
Address Register Indirect with Postincrement	011	register number
Address Register Indirect with Predecrement	100	register number
Address Register Indirect with Displacement	101	register number
Address Register Indirect with Index	110	register number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate or Status Register	111	100

TABLE 5 - IMPLICIT INSTRUCTION REFERENCE SUMMARY

Instruction	Implied Register(s)
Branch Conditional (B_{CC}), Branch Always (BRA)	PC
Branch to Subroutine (BSR)	PC, SP
Check Register against Bounds (CHK)	SSP, SR
Test Condition, Decrement and Branch (DB_{CC})	PC
Signed Divide (DIVS)	SSP, SR
Unsigned Divide (DIVU)	SSP, SR
Jump (JMP)	PC
Jump to Subroutine (JSR)	PC, SP
Link and Allocate (LINK)	SP
Move Condition Codes (MOVE CCR)	SR
Move Status Register (MOVE SR)	SR
Move User Stack Pointer (MOVE USP)	USP
Push Effective Address (PEA)	SP
Return from Exception (RTE)	PC, SP, SR
Return and Restore Condition Codes (RTR)	PC, SP, SR
Return from Subroutine (RTS)	PC, SP
Trap (TRAP)	SSP, SR
Trap on Overflow (TRAPV)	SSP, SR
Unlink (UNLK)	SP

SYSTEM STACK. The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through the addressing modes. Address register seven (A7) is the system stack pointer (SP). The system stack pointer is either the supervisor stack pointer (SSP) or the user stack pointer (USP), depending on the state of the S-bit in the status register. If the S-bit indicates supervisor state, the SSP is the active system stack pointer, and the USP cannot be referenced as an address register. If the S-bit indicates user state, the USP is the active system stack pointer, and the SSP cannot be referenced. Each system stack fills from high memory to low memory.

INSTRUCTION SET SUMMARY

The following paragraphs contain an overview of the form and structure of the MC68000 instruction set. The instructions form a set of tools that include all the machine functions to perform the following operations:

Data Movement
Integer Arithmetic
Shifts and Rotates
Bit Manipulation
Binary Coded Decimal
Program Control
System Control

The complete range of instruction capabilities combined with the flexible addressing modes described previously provide a very flexible base for program development.

DATA MOVEMENT OPERATIONS

The basic method of data acquisition (transfer and storage) is provided by the move (MOVE) instruction. The move instruction and the effective addressing modes allow both address and data manipulation. Data move instructions allow byte, word, and long operands to be transferred from memory to memory, memory to register, register to memory, and register to register. Address move instructions allow word and long operand transfers and ensure that only legal address manipulations are executed. In addition to the general move instruction there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), unlink stack (UNLK), and move quick (MOVEQ). Table 6 is a summary of the data movement operations.

TABLE 6 — DATA MOVEMENT OPERATIONS

Instruction	Operand Size	Operation
EXG	32	Rx↔Ry
LEA	32	EA →An
LINK	—	An →SP@- SP →An SP + d →SP
MOVE	8, 16, 32	(EA)s →EAd
MOVEM	16, 32	(EA) →An, Dn An, Dn →EA
MOVEP	16, 32	(EA) →Dn Dn →EA
MOVEQ	8	#xxx →Dn
PEA	32	EA →SP@-
SWAP	32	Dn [31:16]↔Dn [15:0]
UNLK	—	An →Sp SP @+ →An

Notes:
s = source
d = destination
[ ] = bit numbers
@- = indirect with predecrement
@+ = indirect with postdecrement

INTEGER ARITHMETIC OPERATIONS

The arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP), clear (CLR), and negate (NEG). The add and subtract instructions are available for both address and data operations, with data operations accepting all operand sizes. Address operations are limited to legal address size operands (16 or 32 bits). Data, address, and memory compare operations are also available. The clear and negate instructions may be used on all sizes of data operands.

The multiply and divide operations are available for signed and unsigned operands using word multiply to produce a long product, and a long word dividend with word divisor to produce a word quotient with a word remainder.

Multiprecision and mixed size arithmetic can be accomplished using a set of extended instructions. These instructions are: add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX).

TABLE 7 — INTEGER ARITHMETIC OPERATIONS

Instruction	Operand Size	Operation
ADD	8, 16, 32 16, 32	Dn + (EA) →Dn (EA) + Dn →EA (EA) + #xxx →EA An + (EA) →An
ADDX	8, 16, 32 16, 32	Dx + Dy + X →Dx Ax@- Ay@- + X →Ax@
CLR	8, 16, 32	0 →EA
CMP	8, 16, 32 16, 32	Dn - (EA) (EA) - #xxx Ax@+ - Ay@+ An - (EA)
DIVS	32 ÷ 16	Dn/(EA) →Dn
DIVU	32 ÷ 16	Dn/(EA) →Dn
EXT	8 →16 16 →32	(Dn)8 →Dn16 (Dn)16 →Dn32
MULS	16 * 16 →32	Dn * (EA) →Dn
MULU	16 * 16 →32	Dn * (EA) →Dn
NEG	8, 16, 32	0 - (EA) →EA
NEGX	8, 16, 32	0 - (EA) - X - EA
SUB	8, 16, 32 16, 32	Dn - (EA) →Dn (EA) - Dn →EA (EA) - #xxx →EA An - (EA) →An
SUBX	8, 16, 32	Dx - Dy - X →Dx Ax@- - Ay@- - X →Ax@
TAS	8	(EA) - 0, 1 →EA[7]
TST	8, 16, 32	(EA) - 0

Note:
[ ] = bit number

A test operand (TST) instruction that will set the condition codes as a result of a compare of the operand with zero is also available. Test and set (TAS) is a synchronization instruction useful in multiprocessor systems. Table 7 is a summary of the integer arithmetic operations.

LOGICAL OPERATIONS

Logical operation instructions AND, OR, EOR, and NOT are available for all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. Table 8 is a summary of the logical operations.

TABLE 8 — LOGICAL OPERATIONS

Instruction	Operand Size	Operation
AND	8, 16, 32	Dn ∧ (EA) →Dn (EA) ∧ Dn →EA (EA) ∧ #xxx →EA
OR	8, 16, 32	Dn v (EA) →Dn (EA) v Dn →EA (EA) v #xxx →EA
EOR	8, 16, 32	(EA) ⊕ Dy →EA (EA) ⊕ #xxx →EA
NOT	8, 16, 32	~(EA) →EA

Note: ~ = invert

SHIFT AND ROTATE OPERATIONS

Shift operations in both directions are provided by the arithmetic instructions ASR and ASL and logical shift instructions LSR and LSL. The rotate instructions (with and without extend) available are ROXR, ROXL, ROR, and ROL. All shift and rotate operations can be performed in either registers or memory. Register shifts and rotates support all operand sizes and allow a shift count specified in the instruction of one to eight bits, or 0 to 63 specified in a data register

Memory shifts and rotates are for word operands only and allow only single-bit shifts or rotates.

Table 9 is a summary of the shift and rotate operations.

BIT MANIPULATION OPERATIONS

Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). Table 10 is a summary of the bit manipulation operations.

BINARY CODED DECIMAL OPERATIONS

Multiprecision arithmetic operations on binary coded decimal numbers are accomplished using the following instructions: add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD). Table 11 is a summary of the binary coded decimal operations.

TABLE 9 — SHIFT AND ROTATE OPERATIONS

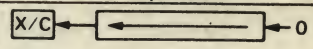
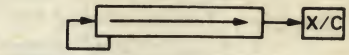
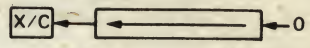
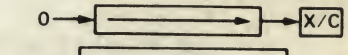
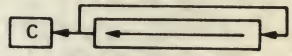
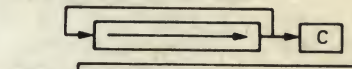
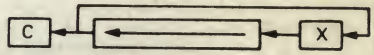
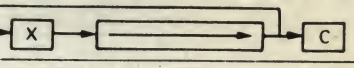
Instruction	Operand Size	Operation
ASL	8, 16, 32	
ASR	8, 16, 32	
LSL	8, 16, 32	
LSR	8, 16, 32	
ROL	8, 16, 32	
ROR	8, 16, 32	
ROXL	8, 16, 32	
ROXR	8, 16, 32	

TABLE 10 — BIT MANIPULATION OPERATIONS

Instruction	Operand Size	Operation
BTST	8, 32	~ bit of (EA) →Z
BSET	8, 32	~bit of (EA) →Z 1 →bit of EA
BCLR	8, 32	~ bit of (EA) →Z 0 →bit of EA
BCHG	8, 32	~ bit of (EA) →Z ~ bit of (EA) → bit of EA

TABLE 11 — BINARY CODED DECIMAL OPERATIONS

Instruction	Operand Size	Operation
ABCD	8	$Dx_{10} + Dy_{10} + X \rightarrow Dx$ $Ax@_{-10} + Ay@_{-10} + X \rightarrow Ax@$
SBCD	8	$Dx_{10} - Dy_{10} - X \rightarrow Dx$ $Ax@_{-10} - Ay@_{-10} - X \rightarrow Ax@$
NBCD	8	$0 - (EA)_{10} - X \rightarrow EA$

PROGRAM CONTROL OPERATIONS

Program control operations are accomplished using a series of conditional and unconditional branch instructions and return instructions. These instructions are summarized in Table 12.

The conditional instructions provide setting and branching for the following conditions:

CC	— carry clear	LS	— low or same
CS	— carry set	LT	— less than
EQ	— equal	MI	— minus
F	— never true	NE	— not equal
GE	— greater or equal	PL	— plus
GT	— greater than	T	— always true
HI	— high	VC	— no overflow
LE	— less or equal	VS	— overflow

TABLE 12 — PROGRAM CONTROL OPERATIONS

Instruction	Operation
Conditional	
Bcc	Branch conditionally (14 conditions) 8- and 16-bit displacement
DBcc	Test condition, decrement, and branch. 16-bit displacement
Scc	Set byte conditionally (16 conditions)
Unconditional	
BRA	Branch always 8- and 16-bit displacement
BSR	Branch to subroutine 8- and 16-bit displacement
JMP	Jump
JSR	Jump to subroutine
Returns	
RTR	Return and restore condition codes
RTS	Return from subroutine

TABLE 13 — SYSTEM CONTROL OPERATIONS

Instruction	Operation
Privileged	
RESET	Reset external devices
RTE	Return from exception
STOP	Stop program execution
ORI to SR	Logical OR to status register
MOVE USP	Move user stack pointer
ANDI to SR	Logical AND to status register
EORI to SR	Logical EOR to status register
MOVE EA to SR	Load new status register
Trap Generating	
TRAP	Trap
TRAPV	Trap on overflow
CHK	Check register against bounds
Status Register	
ANDI to CCR	Logical AND to condition codes
EORI to CCR	Logical EOR to condition codes
MOVE EA to CCR	Load new condition codes
ORI to CCR	Logical OR to condition codes
MOVE SR to EA	Store status register

SYSTEM CONTROL OPERATIONS

System control operations are accomplished by using privileged instructions, trap generating instructions, and instructions that use or modify the status register. These instructions are summarized in Table 13.

SIGNAL AND BUS OPERATION DESCRIPTION

The following paragraphs contain a brief description of the input and output signals. A discussion of bus operation during the various machine cycles and operations is also given.

SIGNAL DESCRIPTION

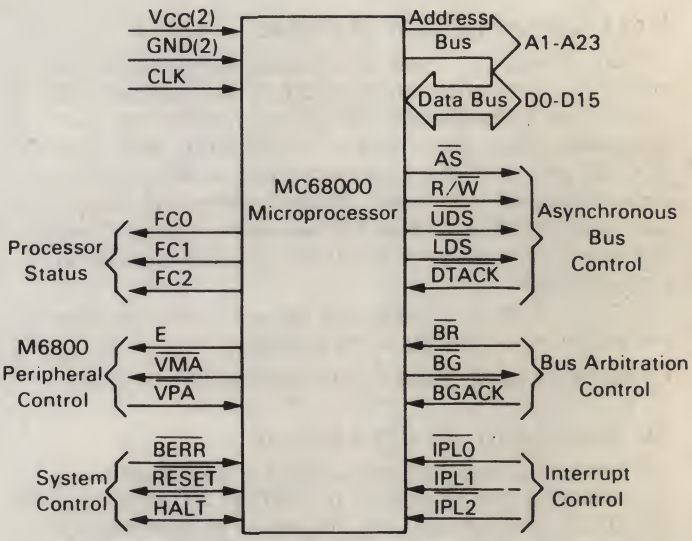
The input and output signals can be functionally organized into the groups shown in Figure 9. The following paragraphs provide a brief description of the signals and also a reference (if applicable) to other chapters that contain more detail about the function being performed.

ADDRESS BUS (A1 THROUGH A23). This 23-bit, unidirectional, three-state bus is capable of addressing 8 megawords of data. It provides the address for bus operation during all cycles except interrupt cycles. During interrupt cycles, address lines A1, A2, and A3 provide information about what level interrupt is being serviced while address lines A4 through A23 are all set to a logic high.

DATA BUS (D0 THROUGH D15). This 16-bit, bidirectional, three-state bus is the general purpose data path. It can transfer and accept data in either word or byte length. During an interrupt acknowledge cycle, the external device supplies the vector number on data lines D0-D7.

ASYNCHRONOUS BUS CONTROL. Asynchronous data transfers are handled using the following control signals: address strobe, read/write, upper and lower data strobes, and data transfer acknowledge. These signals are explained in the following paragraphs.

FIGURE 9 — INPUT AND OUTPUT SIGNALS



Address Strobe ($\overline{AS}$). This signal indicates that there is a valid address on the address bus.

Read/Write (R/$\overline{W}$). This signal defines the data bus transfer as a read or write cycle. The R/$\overline{W}$ signal also works in conjunction with the upper and lower data strobes as explained in the following paragraph.

Upper And Lower Data Strobes ($\overline{UDS}$, $\overline{LDS}$). These signals control the data on the data bus, as shown in Table 14. When the R/$\overline{W}$ line is high, the processor will read from the data bus as indicated. When the R/$\overline{W}$ line is low, the processor will write to the data bus as shown.

TABLE 14 — DATA STROBE CONTROL OF DATA BUS

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	D8-D15	D0-D7
High	High	—	No valid data	No valid data
Low	Low	High	Valid data bits 8-15	Valid data bits 0-7
High	Low	High	No valid data	Valid data bits 0-7
Low	High	High	Valid data bits 8-15	No valid data
Low	Low	Low	Valid data bits 8-15	Valid data bits 0-7
High	Low	Low	Valid data bits 0-7*	Valid data bits 0-7
Low	High	Low	Valid data bits 8-15	Valid data bits 8-15*

*These conditions are a result of current implementation and may not appear on future devices.

Data Transfer Acknowledge ($\overline{DTACK}$). This input indicates that the data transfer is completed. When the processor recognizes $\overline{DTACK}$ during a read cycle, data is latched and the bus cycle terminated. When $\overline{DTACK}$ is recognized during a write cycle, the bus cycle is terminated.

BUS ARBITRATION CONTROL. These three signals form a bus arbitration circuit to determine what device will be the bus master device.

Bus Request ($\overline{BR}$). This input is wire ORed with all other devices that could be bus masters. This input indicates to the processor that some other device desires to become the bus master.

Bus Grant ($\overline{BG}$). This output indicates to all other potential bus master devices that the processor will release bus control at the end of the current bus cycle.

Bus Grant Acknowledge ($\overline{BGACK}$). This input indicates that some other device has become the bus master. This signal cannot be asserted until the following four conditions are met:

1. a bus grant has been received
2. address strobe is inactive which indicates that the microprocessor is not using the bus
3. data transfer acknowledge is inactive which indicates that either memory or the peripherals are not using the bus
4. bus grant acknowledge is inactive which indicates that no other device is still claiming bus mastership.

INTERRUPT CONTROL ($\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$). These input pins indicate the encoded priority level of the device requesting an interrupt. Level seven is the highest priority while level zero indicates that no interrupts are requested. The least significant bit is given in $\overline{IPL0}$ and the most significant bit is contained in $\overline{IPL2}$.

SYSTEM CONTROL. The system control inputs are used to either reset or halt the processor and to indicate to the processor that bus errors have occurred. The three system control inputs are explained in the following paragraphs.

Bus Error ($\overline{BERR}$). This input informs the processor that there is a problem with the cycle currently being executed. Problems may be a result of:

1. nonresponding devices
2. interrupt vector number acquisition failure
3. illegal access request as determined by a memory management unit
4. other application dependent errors.

The bus error signal interacts with the halt signal to determine if exception processing should be performed or the current bus cycle should be retried.

Refer to **BUS ERROR AND HALT OPERATION** paragraph for additional information about the interaction of the bus error and halt signals.

Reset ($\overline{RESET}$). This bidirectional signal line acts to reset (initiate a system initialization sequence) the processor in response to an external reset signal. An internally generated reset (result of a RESET instruction) causes all external devices to be reset and the internal state of the processor is not affected. A total system reset (processor and external devices) is the result of external halt and reset signals applied at the same time. Refer to **RESET OPERATION** paragraph for additional information about reset operation.

Halt ($\overline{HALT}$). When this bidirectional line is driven by an external device, it will cause the processor to stop at the completion of the current bus cycle. When the processor has been halted using this input, all control signals are inactive and all three-state lines are put in their high-impedance state. Refer to **BUS ERROR AND HALT OPERATION** paragraph for additional information about the interaction between the halt and bus error signals.

When the processor has stopped executing instructions, such as in a double bus fault condition, the halt line is driven by the processor to indicate to external devices that the processor has stopped.

M6800 PERIPHERAL CONTROL. These control signals are used to allow the interfacing of synchronous M6800 peripheral devices with the asynchronous MC68000. These signals are explained in the following paragraphs.

Enable (E). This signal is the standard enable signal common to all M6800 type peripheral devices. The period for this output is ten MC68000 clock periods (six clocks low; four clocks high).

Valid Peripheral Address ($\overline{VPA}$). This input indicates that the device or region addressed is a M6800 family device and that data transfer should coincide with the enable (E) signal. This input also indicates that the processor should use automatic vectoring for an interrupt. Refer to **INTERFACE WITH M6800 PERIPHERALS**.

Valid Memory Address ($\overline{VMA}$). This output is used to indicate to M6800 peripheral devices that there is a valid address on the address bus and the processor is synchronized to enable. This signal only responds to a valid peripheral address ($\overline{VPA}$) input which indicates that the peripheral is a M6800 family device.

PROCESSOR STATUS (FC0, FC1, FC2). These function code outputs indicate the mode (user or supervisor) and the cycle type currently being executed, as shown in Table 15. The information indicated by the function code outputs is valid whenever address strobe ($\overline{AS}$) is active.

TABLE 15 — FUNCTION CODE OUTPUTS

FC2	FC1	FC0	Cycle Type
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

CLOCK (CLK). The clock input is a TTL compatible signal that is internally buffered for development of the internal clocks needed by the processor. The clock input shall be a constant frequency.

SIGNAL SUMMARY. Table 16 is a summary of all the signals discussed in the previous paragraphs.

TABLE 16 — SIGNAL SUMMARY

Signal Name	Mnemonic	Input/Output	Active State	Three State
Address Bus	A1-A23	output	high	yes
Data Bus	D0-D15	input/output	high	yes
Address Strobe	$\overline{AS}$	output	low	yes
Read/Write	R/$\overline{W}$	output	read-high write-low	yes
Upper and Lower Data Strobes	$\overline{UDS}$, $\overline{LDS}$	output	low	yes
Data Transfer Acknowledge	$\overline{DTACK}$	input	low	no
Bus Request	$\overline{BR}$	input	low	no
Bus Grant	$\overline{BG}$	output	low	no
Bus Grant Acknowledge	$\overline{BGACK}$	input	low	no
Interrupt Priority Level	$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	input	low	no
Bus Error	$\overline{BERR}$	input	low	no
Reset	$\overline{RESET}$	input/output	low	no*
Halt	$\overline{HALT}$	input/output	low	no*
Enable	E	output	high	no
Valid Memory Address	$\overline{VMA}$	output	low	yes
Valid Peripheral Address	$\overline{VPA}$	input	low	no
Function Code Output	FC0, FC1, FC2	output	high	yes
Clock	CLK	input	high	no
Power Input	V_{CC}	Input	—	—
Ground	GND	input	—	—

*open drain

BUS OPERATION

The following paragraphs explain control signal and bus operation during data transfer operations, bus arbitration, bus error and halt conditions, and reset operation.

DATA TRANSFER OPERATIONS. Transfer of data between devices involves the following leads:

- Address Bus A1 through A23
- Data Bus D0 through D15
- Control Signals

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus structure. In all cycles, the bus master assumes responsibility for deskewing all signals it issues at both the start and end of a cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave device.

The following paragraphs explain the read, write, and read-modify-write cycles. The indivisible read-modify-write cycle is the method used by the MC68000 for interlocked multiprocessor communications.

NOTE

The terms **assertion** and **negation** will be used extensively. This is done to avoid confusion when dealing with a mixture of "active-low" and "active-high" signals. The term assert or assertion is used to indicate that a signal is active or true independent of whether that voltage is low or high. The term negate or negation is used to indicate that a signal is inactive or false.

Read Cycle. During a read cycle, the processor receives data from memory or a peripheral device. The processor reads bytes of data in all cases. If the instruction specifies a word (or double word) operation, the processor reads both bytes. When the instruction specifies byte operation, the processor uses an internal A0 bit to determine which byte to read and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. When the data is received, the processor correctly positions it internally.

A word read cycle flow chart is given in Figure 10. A byte read cycle flow chart is given in Figure 11. Read cycle timing is given in Figure 12 and Figure 13 details word and byte read cycle operation.

FIGURE 10 — WORD READ CYCLE FLOW CHART

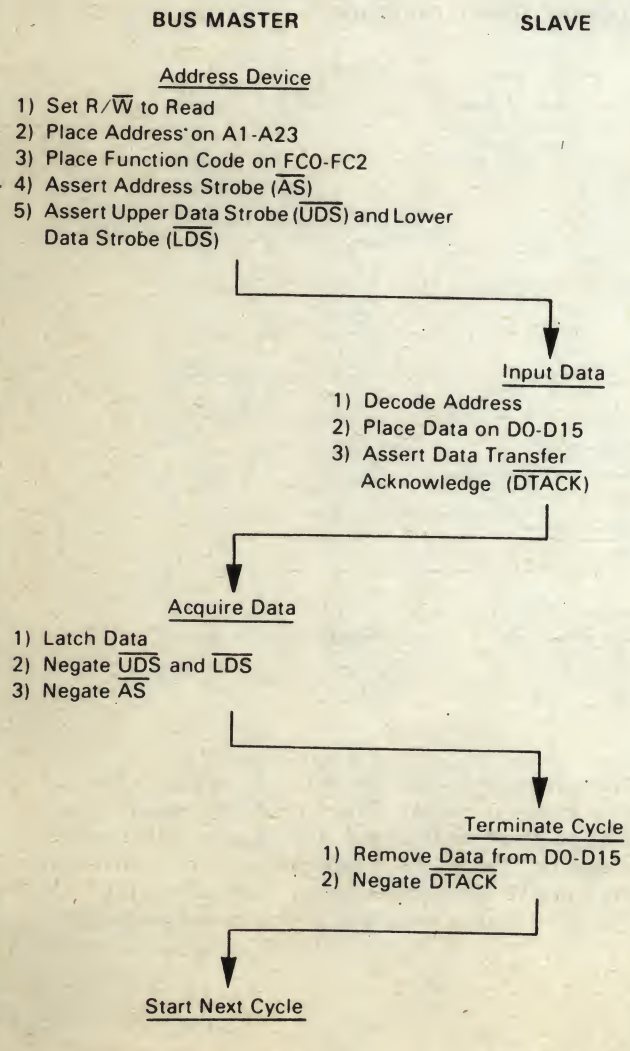


FIGURE 11 — BYTE READ CYCLE FLOW CHART

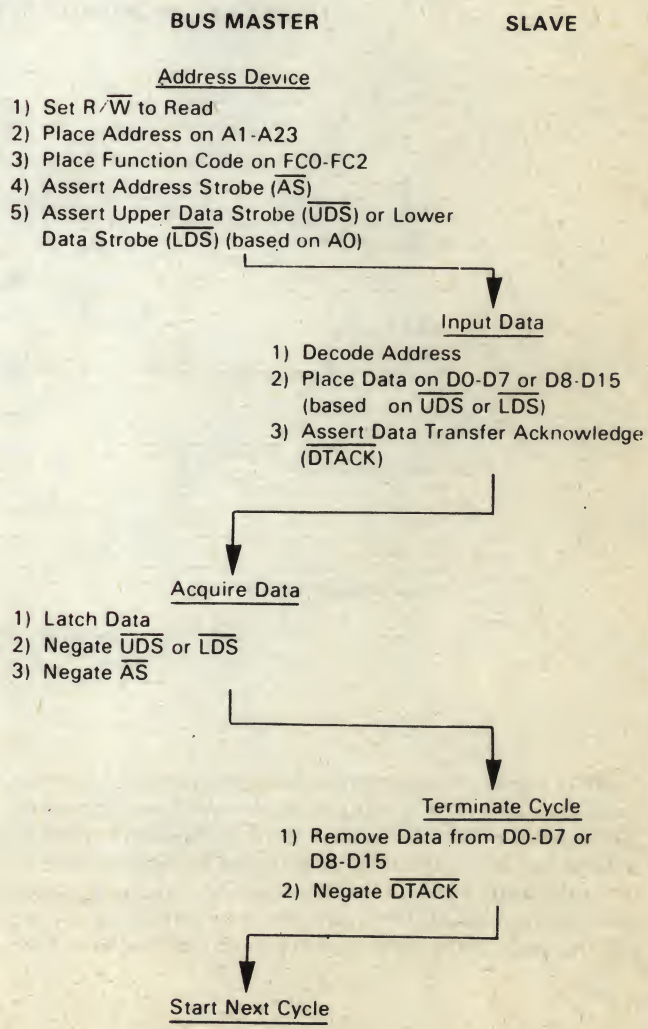


FIGURE 12 — READ AND WRITE CYCLE TIMING DIAGRAM

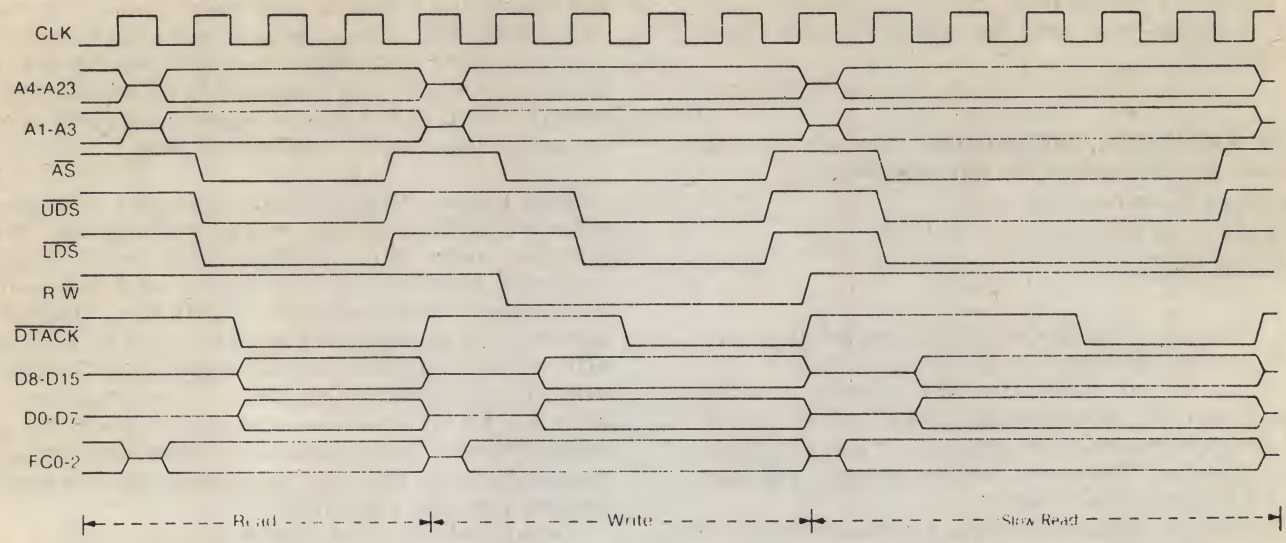
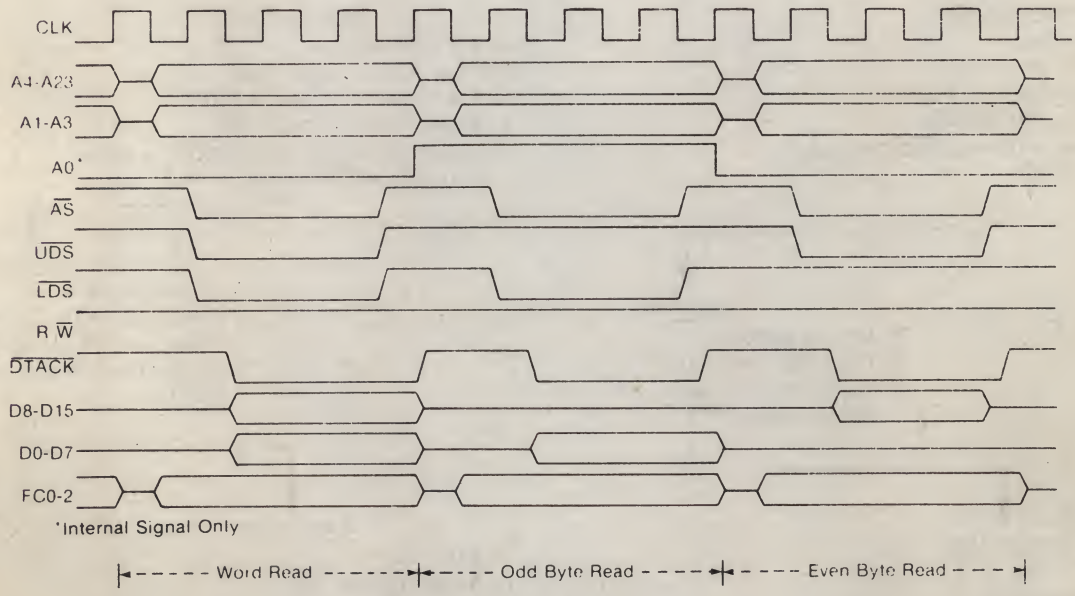


FIGURE 13 -- WORD AND BYTE READ CYCLE TIMING DIAGRAM



Write Cycle. During a write cycle, the processor sends data to memory or a peripheral device. The processor writes bytes of data in all cases. If the instruction specifies a word operation, the processor writes both bytes. When the instruction specifies a byte operation, the processor uses an internal A0 bit to determine which byte to write and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. A word write cycle flow chart is given in Figure 14. A byte write cycle flow chart is given in Figure 15. Write cycle timing is given in Figure 12 and Figure 16 details word and byte write cycle operation.

FIGURE 14 — WORD WRITE CYCLE FLOW CHART

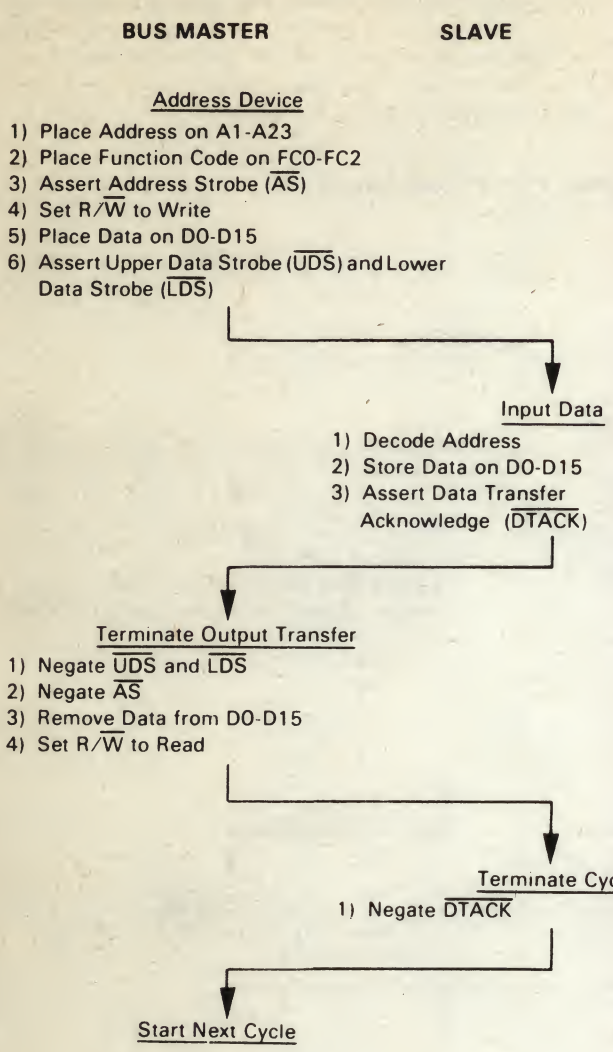


FIGURE 15 — BYTE WRITE CYCLE FLOW CHART

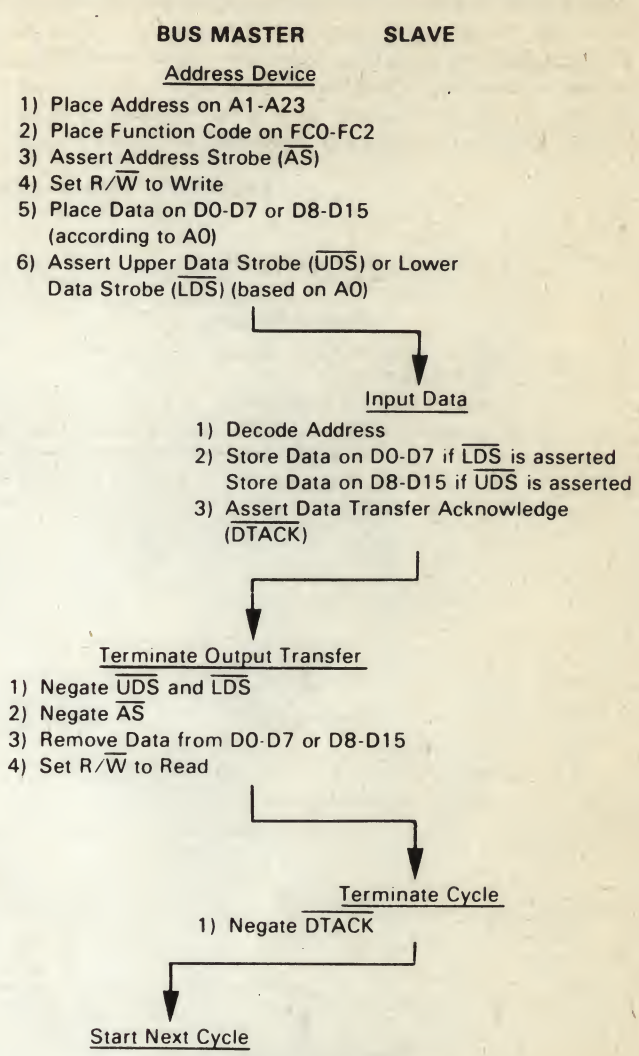
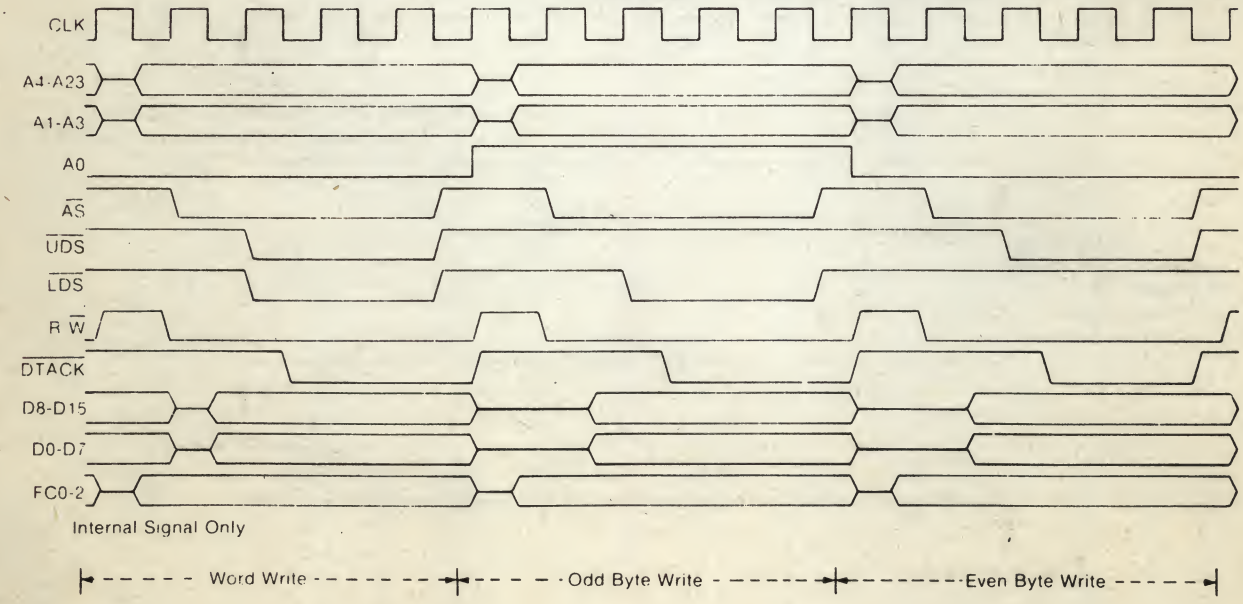


FIGURE 16 — WORD AND BYTE WRITE CYCLE TIMING DIAGRAM



Read-Modify-Write Cycle. The read-modify-write cycle performs a read, modifies the data in the arithmetic-logic unit, and writes the data back to the same address. In the MC68000 this cycle is indivisible in that the address strobe is asserted throughout the entire cycle. The test and set (TAS) instruction uses this cycle to provide meaningful communication between processors in a multiple processor environment. This instruction is the only instruction that uses the read-modify-write cycle and since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. A read-modify-write cycle flow chart is given in Figure 17 and a timing diagram is given in Figure 18.

FIGURE 17 — READ-MODIFY-WRITE CYCLE FLOW CHART

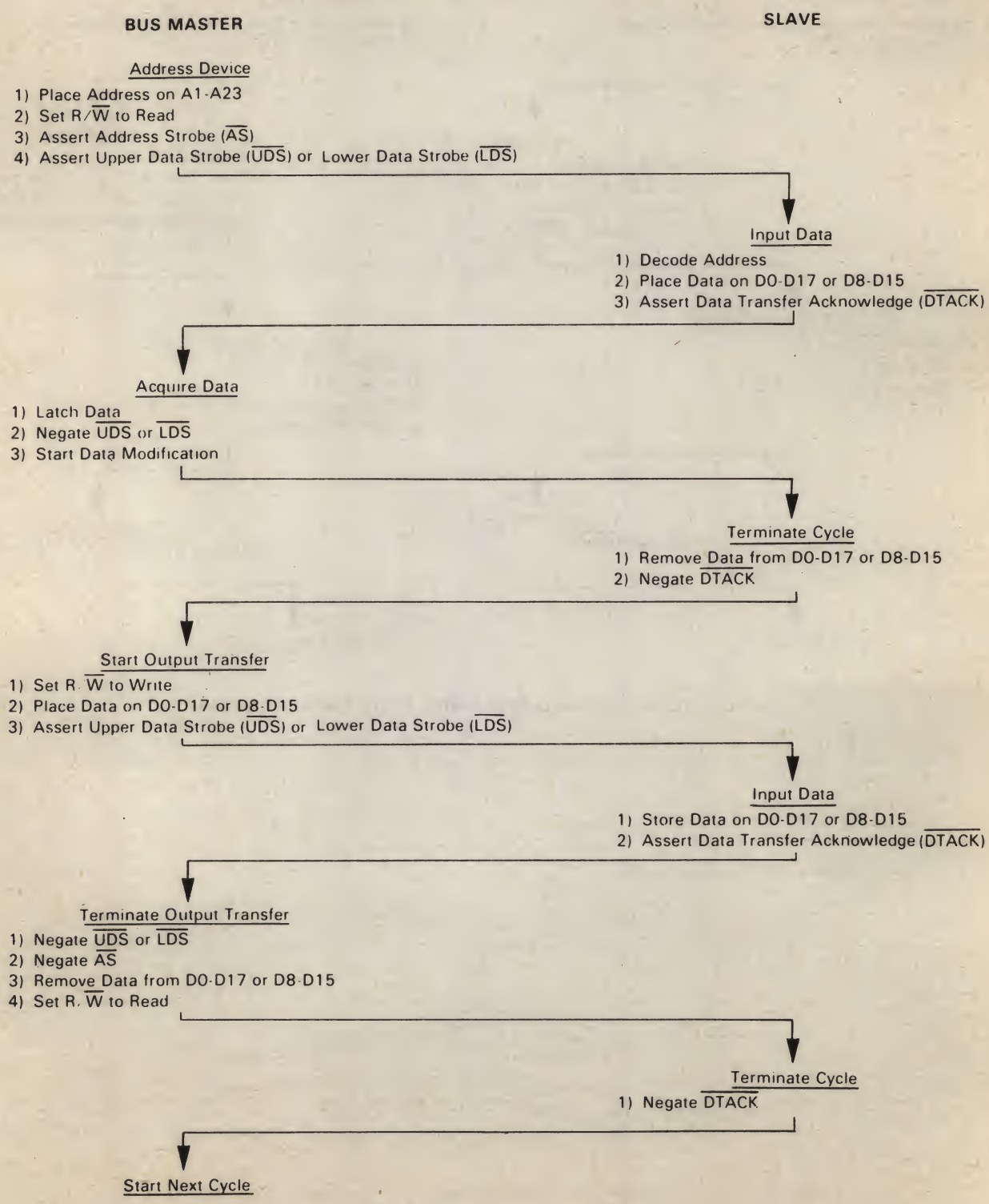
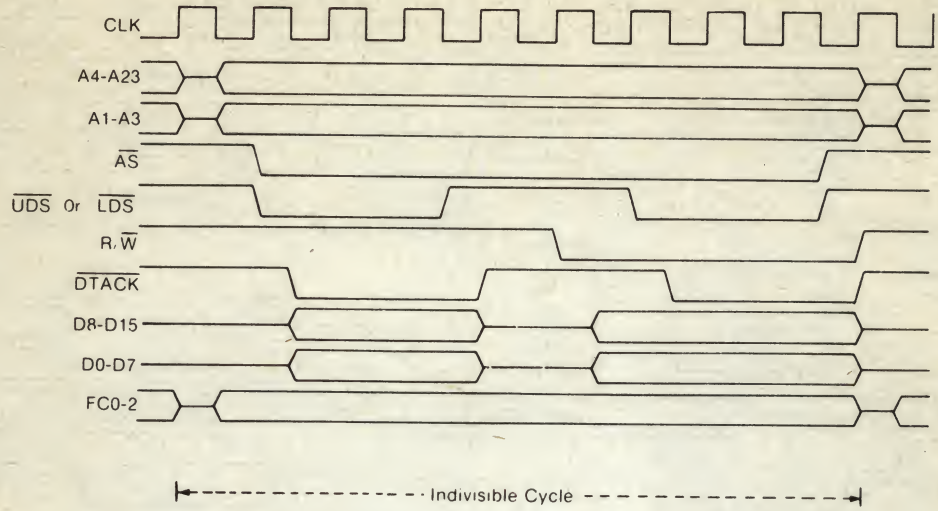


FIGURE 18 — READ-MODIFY-WRITE CYCLE TIMING DIAGRAM



BUS ARBITRATION. Bus arbitration is a technique used by master type devices to request, be granted, and acknowledge bus mastership. In its simplest form, it consists of:

1. Asserting a bus mastership request.
2. Receiving a grant that the bus is available at the end of the current cycle.
3. Acknowledging that mastership has been assumed.

Figure 19 is a flow chart showing the detail involved in a request from a single device. Figure 20 is a timing diagram for the same operations. This technique allows processing of bus requests during data transfer cycles.

The timing diagram shows that the bus request is negated at the time that an acknowledge is asserted. This type of operation would be true for a system consisting of the processor and one device capable of bus mastership. In systems having a number of devices capable of bus mastership, the bus request line from each device is wire ORed to the processor. In this system, it is easy to see that there could be more than one bus request being made. The timing diagram shows that the bus grant signal is negated a few clock cycles after the transition of the acknowledge ($\overline{BGACK}$) signal.

However, if the bus requests are still pending, the processor will assert another bus grant within a few clock cycles after it was negated. This additional assertion of bus grant allows external arbitration circuitry to select the next bus master before the current bus master has completed its requirements. The following paragraphs provide additional information about the three steps in the arbitration process.

FIGURE 19 — BUS ARBITRATION CYCLE FLOW-CHART

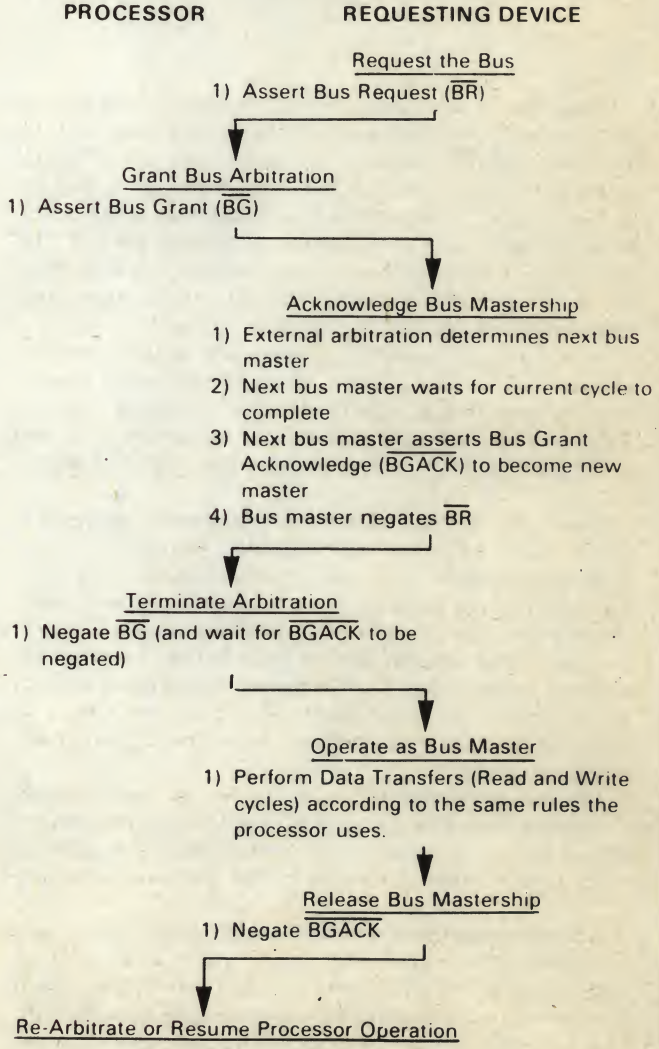
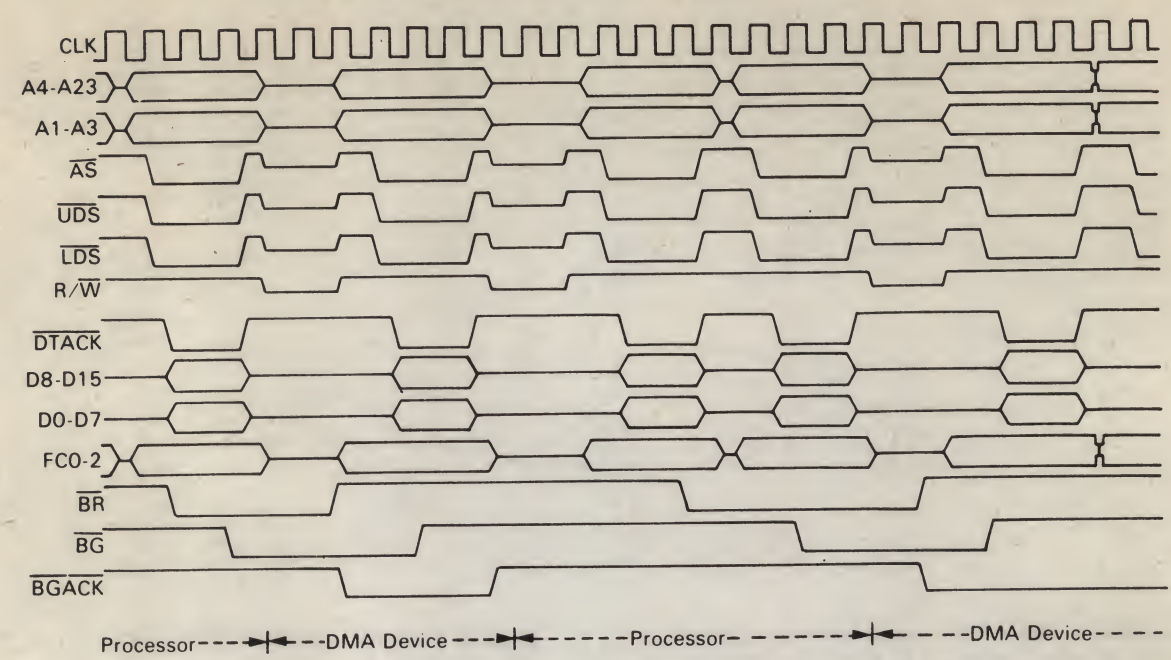


FIGURE 20 — BUS ARBITRATION CYCLE TIMING DIAGRAM



Requesting the Bus. External devices capable of becoming bus masters request the bus by asserting the bus request ($\overline{BR}$) signal. This is a wire ORed signal (although it need not be constructed from open collector devices) that indicates to the processor that some external device requires control of the external bus. The processor is effectively at a lower bus priority level than the external device and will relinquish the bus after it has completed the last bus cycle it has started.

When no acknowledge is received before the bus request signal goes inactive, the processor will continue processing when it detects that bus request is inactive. This allows ordinary processing to continue if the arbitration circuitry responded to noise inadvertently.

Receiving the Bus Grant. The processor issues bus grant ($\overline{BG}$) as soon as possible. Normally this is immediately after internal synchronization. The only exception to this occurs when the processor has made an internal decision to execute the next bus cycle but has not progressed far enough into the cycle to have asserted the address strobe ($\overline{AS}$) signal. In this case, bus grant will not be asserted until one clock after address strobe is asserted to indicate to external devices that a bus cycle is being executed.

The bus grant signal may be routed through a daisy-chained network or through a specific priority-encoded network. The processor is not affected by the external method of arbitration as long as the protocol is obeyed.

Acknowledgement of Mastership. Upon receiving a bus grant, the requesting device waits until address strobe, data transfer acknowledge, and bus grant acknowledge are negated before issuing its own $\overline{BGACK}$. The negation of the address strobe indicates that the previous master has completed its cycle, the negation of bus grant acknowledge indicates that the previous master has released the bus. (While address strobe is asserted no device is allowed to "break into" a cycle). The negation of data transfer acknowledge indicates the previous slave has terminated its connection to the previous master. Note that in some applications data transfer acknowledge might not enter into this function. General purpose devices would then be connected such that they were only dependent on address strobe. When bus grant acknowledge is issued the device is bus master until it negates bus grant acknowledge. Bus grant acknowledge should not be negated until after the bus cycle(s) is (are) completed. Bus mastership is terminated at the negation of bus grant acknowledge.

The bus request from the granted device should be dropped when bus grant acknowledge is asserted. If bus request is still asserted after bus grant acknowledge is negated the processor performs another arbitration sequence and issues another bus grant. Note that the processor does not perform any external bus cycles before it re-asserts bus grant.

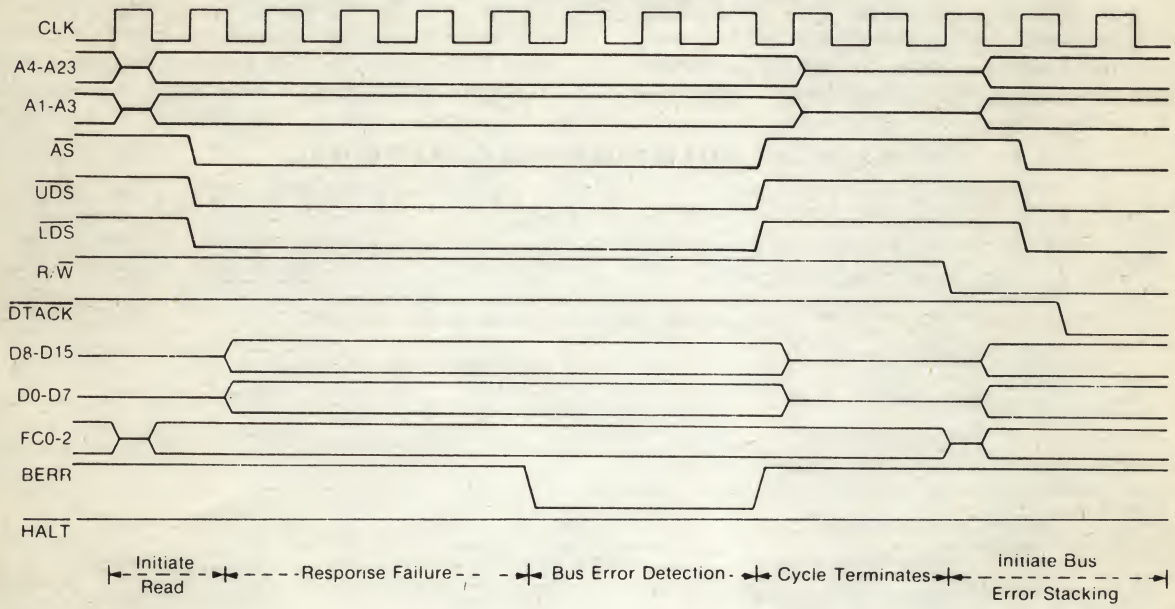
BUS ERROR AND HALT OPERATION. In a bus architecture that requires a handshake from an external device, the possibility exists that the handshake might not occur. Since different systems will require a different maximum response time, a bus error input is provided. External circuitry must be used to determine the duration between address strobe and data transfer acknowledge before issuing a bus error signal. When a bus error signal is received, the processor has two options: initiate a bus error exception sequence or try running the bus cycle again.

Exception Sequence. The bus error exception sequence is entered when the processor receives a bus error signal and the halt pin is inactive. Figure 21 is a timing diagram for the exception sequence. The sequence is composed of the following elements:

1. Stacking the program counter and status register
2. Stacking the error information
3. Reading the bus error vector table entry
4. Executing the bus error handler routine

The stacking of the program counter and the status register is the same as if an interrupt had occurred. Several additional items are stacked when a bus error occurs. These items are used to determine the nature of the error and to correct it, if possible. The bus error vector is vector number two located at address $000008. The processor loads the new program counter from this location. A software bus error handler routine is then executed by the processor. Refer to **EXCEPTION PROCESSING** for additional information.

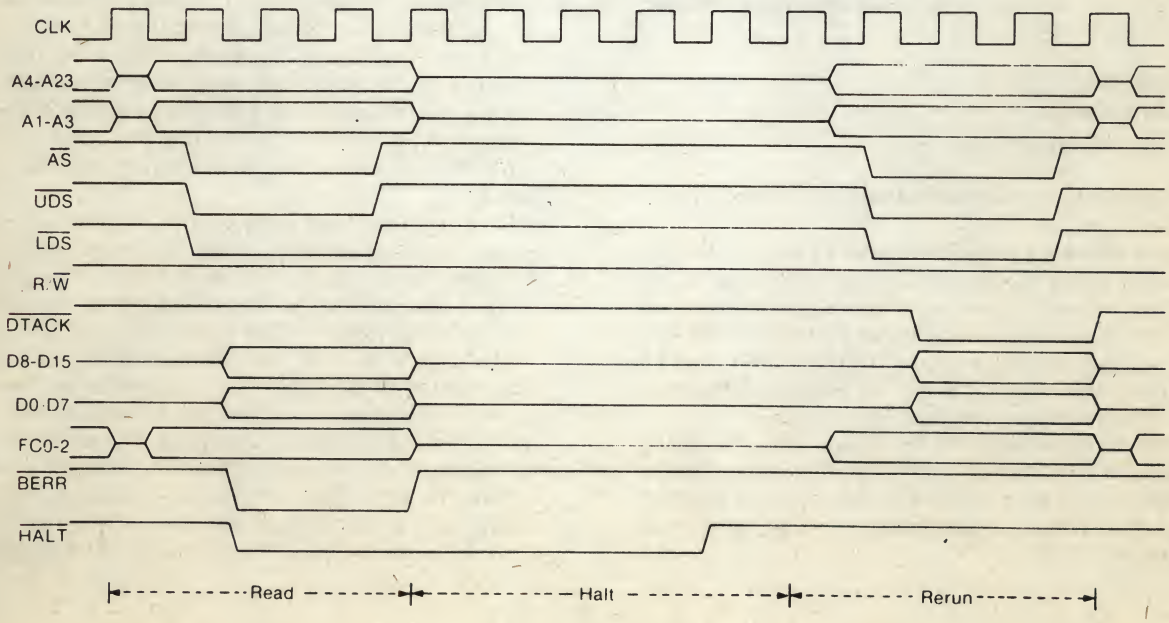
FIGURE 21 — BUS ERROR TIMING DIAGRAM



Re-Running the Bus Cycle. When the processor receives a bus error signal and the halt pin is being driven by an external device, the processor enters the re-run sequence. Figure 22 is a timing diagram for re-running the bus cycle.

The processor completes the bus cycle, then puts the address, data, function code, and control leads in the high-impedance state. The processor remains "halted," and will not run another bus cycle until the halt signal is removed by external logic. Then the processor will re-run

FIGURE 22 — RE-RUN BUS CYCLE TIMING INFORMATION



the previous bus cycle using the same address, the same function codes, the same data (for a write operation), and the same controls. The bus error signal should be removed before the halt signal is removed.

NOTE

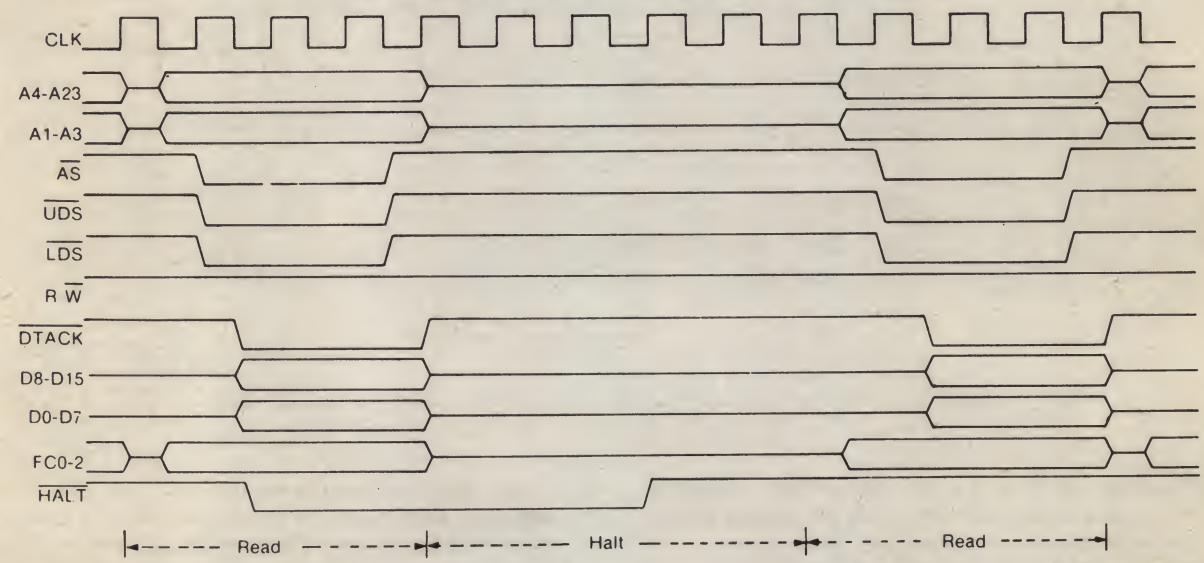
The processor will not re-run a read-modify-write cycle. This restriction is made to guarantee that the entire cycle runs correctly and that the write operation of a Test-and-Set operation is performed without ever releasing AS.

Halt Operation with No Bus Error. The halt input signal to the MC68000 performs a Halt/Run/Single-Step function in a similar fashion to the M6800 halt function. The halt and run modes are somewhat self explanatory in that when the halt signal is constantly active the processor "halts" (does nothing) and when the halt signal is constantly inactive the processor "runs" (does something).

The single-step mode is derived from correctly timed transitions on the halt signal input. It forces the processor to execute a single bus cycle by entering the "run" mode until the processor starts a bus cycle then changing to the "halt" mode. Thus, the single step mode allows the user to proceed through (and therefore debug) processor operations one bus cycle at a time.

Figure 23 details the timing required for correct single-step operations. Some care must be exercised to avoid harmful interactions between the bus error signal and the halt pin when using the single cycle mode as a debugging tool. This is also true of interactions between the halt and reset lines since these can reset the machine.

FIGURE 23 — HALT SIGNAL TIMING CHARACTERISTICS



When the processor completes a bus cycle after recognizing that the halt signal is active, most three-state signals are put in the high-impedance state. These include:

1. address lines
2. data lines
3. function code lines

This is required for correct performance of the re-run bus cycle operation.

Note that when the processor honors a request to halt, the function codes are put in the high-impedance state (their buffer characteristics are the same as the address buffers). While the processor is honoring the halt request, bus arbitration performs as usual. That is, halting has no effect on bus arbitration. It is the bus arbitration function that removes the control signals from the bus.

The halt function and the hardware trace capability allow the hardware debugger to trace single bus cycles or single instructions at a time. These processor capabilities, along with a software debugging package, give total debugging flexibility.

Double Bus Faults. When a bus error exception occurs, the processor will attempt to stack several words containing information about the state of the machine. If a bus error exception occurs during the stacking operation, there have been two bus errors in a row. This is commonly referred to as a double bus fault. When a double bus fault occurs, the processor will halt. Once a bus error exception has occurred, any bus error exception occurring before the execution of the next instruction constitutes a double bus fault.

Note that a bus cycle which is re-run does not constitute a bus error exception, and does not contribute to a double bus fault. Note also that this means that as long as the external hardware requests it, the processor will continue to re-run the same bus cycle.

The bus error pin also has an effect on processor operation after the processor receives an external reset input. The processor reads the vector table after a reset to determine the address to start program execution. If a bus error occurs while reading the vector table (or at any time before the first instruction is executed), the processor reacts as if a double bus fault has occurred and it halts. Only an external reset will start a halted processor.

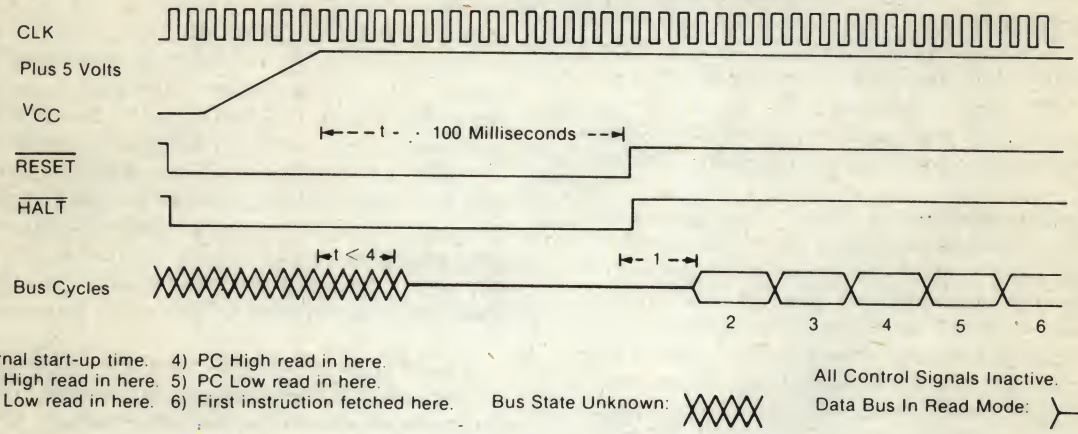
RESET OPERATION. The reset signal is a bidirectional signal that allows either the processor or an external signal to reset the system. Figure 24 is a timing diagram for reset operations. Both the halt and the reset lines must be applied to ensure total reset of the processor.

When the reset and halt lines are driven by an external device, it is recognized as an entire system reset, including the processor. The processor responds by reading the reset vector table entry (vector number zero, address $000000) and loads it into the supervisor stack pointer (SSP). Vector table entry number one at address $000004 is read next and loaded into the program counter. The processor initializes the status register to an interrupt level of seven. No other registers are affected by the reset sequence.

When a RESET instruction is executed, the processor drives the reset pin for 124 clock pulses. In this case, the processor is trying to reset the rest of the system. Therefore, there is no effect on the internal state of the processor. All of the processor's internal registers and the status register are unaffected by the execution of a RESET instruction. All external devices connected to the reset line should be reset at the completion of the RESET instruction.

When V_{CC} is initially applied to the processor, an external reset must be applied to the reset pin for 100 milliseconds.

FIGURE 24 — RESET OPERATION TIMING DIAGRAM



Notes:
1) Internal start-up time.
2) SSP High read in here.
3) SSP Low read in here.
4) PC High read in here.
5) PC Low read in here.
6) First instruction fetched here.

Bus State Unknown:
All Control Signals Inactive.
Data Bus In Read Mode:

EXCEPTION PROCESSING

The following paragraphs describe the actions of the MC68000 which are outside the normal processing associated with the execution of instructions. The functions of the bits in the supervisor portion of the status register are covered: the supervisor/user bit, the trace enable bit, and the processor interrupt priority mask. Finally, the sequence of memory references and actions taken by the processor on exception conditions is detailed.

PROCESSING STATES

The MC68000 is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. A special case of the normal state is the stopped state which the processor enters when a STOP instruction is executed. In this state, no further memory references are made.

The exception processing state is associated with interrupts, trap instructions, tracing and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing is designed to provide an efficient context switch so that the processor may handle unusual conditions.

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.

PRIVILEGE STATES

The processor operates in one of two states of privilege: the "user" state or the "supervisor" state. The privilege state determines which operations are legal, is used by the external memory management device to control and translate accesses, and is used to choose between the supervisor stack pointer and the user stack pointer in instruction references.

The privilege state is a mechanism for providing security in a computer system. Programs should access only their own code and data areas, and ought to be restricted from accessing information which they do not need and must not modify.

The privilege mechanism provides security by allowing most programs to execute in user state. In this state, their accesses are controlled, and their effects on other parts of the system are limited. The operating system executes in the supervisor state, has access to all resources, and performs the overhead tasks for the user state programs.

SUPERVISOR STATE. The supervisor state is the higher state of privilege. For instruction execution, the supervisor state is determined by the S-bit of the status register; if the S-bit is asserted (high), the processor is in the supervisor state. All instructions can be executed in the supervisor state. The bus cycles generated by instructions executed in the supervisor state are classified as supervisor references. While the processor is in the supervisor privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly access the supervisor stack pointer.

All exception processing is done in the supervisor state, regardless of the setting of the S-bit. The bus cycles generated during exception processing are classified as supervisor references. All stacking operations during exception processing use the supervisor stack pointer.

USER STATE. The user state is the lower state of privilege. For instruction execution, the user state is determined by the S-bit of the status register; if the S-bit is negated (low), the processor is executing instructions in the user state.

Most instructions execute the same in user state as in the supervisor state. However, some instructions which have important system effects are made privileged. User programs are not permitted to execute the STOP instruction, or the RESET instruction. To ensure that a user program cannot enter the supervisor state except in a controlled manner, the instructions which modify the whole status register are privileged. To aid in debugging programs which are to be used as operating systems, the move to user stack pointer (MOVE to USP) and move from user stack pointer (MOVE from USP) instructions are also privileged.

The bus cycles generated by an instruction executed in user state are classified as user state references. This allows an external memory management device to translate the address and to control access to protected portions of the address space. While the processor is in the user privilege state, those instructions which use either the system stack pointer implicitly, or address register seven explicitly, access the user stack pointer.

PRIVILEGE STATE CHANGES. Once the processor is in the user state and executing instructions, only exception processing can change the privilege state. During exception processing, the current setting of the S-bit of the status register is saved and the S-bit is asserted, putting the processing in the supervisor state. Therefore, when instruction execution resumes at the address specified to process the exception, the processor is in the supervisor privilege state.

REFERENCE CLASSIFICATION. When the processor makes a reference, it classifies the kind of reference being made, using the encoding on the three function code output lines. This allows external translation of addresses, control of access, and differentation of special processor states, such as interrupt acknowledge. Table 17 lists the classification of references.

TABLE 17 — REFERENCE CLASSIFICATION

Function Code Output			
FC2	FC1	FC0	Reference Class
0	0	0	(Unassigned)
0	0	1	User Data
0	1	0	User Program
0	1	1	(Unassigned)
1	0	0	(Unassigned)
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

EXCEPTION PROCESSING

Before discussing the details of interrupts, traps, and tracing, a general description of exception processing is in order. The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary copy of the status register is made, and the status register is set for exception processing. In the second step the exception vector is determined, and the third step is the saving of the current processor context. In the fourth step a new context is obtained, and the processor switches to instruction processing.

EXCEPTION VECTORS. Exception vectors are memory locations from which the processor fetches the address of a routine which will handle that exception. All exception vectors are two words in length (Figure 25), except for the reset vector, which is four words. All exception vectors lie in the supervisor data space, except for the reset vector which is in the supervisor program space. A vector number is an eight-bit number which, when multiplied by four, gives the address of an exception vector. Vector numbers are generated internally or externally, depending on the cause of the exception. In the case of interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (Figure 26) to the processor on data bus lines D0 through D7. The processor translates the vector number into a full 24-bit address, as shown in Figure 27. The memory layout for exception vectors is given in Table 18.

FIGURE 25 — EXCEPTION VECTOR FORMAT

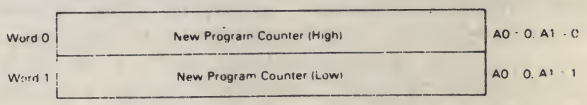
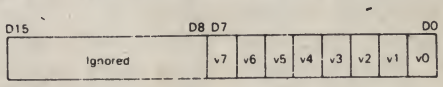
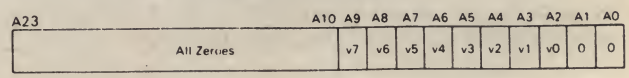


FIGURE 26 — PERIPHERAL VECTOR NUMBER FORMAT



Where
v7 is the MSB of the Vector Number
v0 is the LSB of the Vector Number

FIGURE 27 — ADDRESS TRANSLATED FROM 8-BIT VECTOR NUMBER



As shown in Table 18, the memory layout is 512 words long (1024 bytes). It starts at address 0 and proceeds through address 1023. This provides 255 unique vectors; some of these are reserved for TRAPS and other system functions. Of the 255, there are 192 reserved for user interrupt vectors. However, there is no protection on the first 64 entries, so user interrupt vectors may overlap at the discretion of the systems designer.

KINDS OF EXCEPTIONS. Exceptions can be generated by either internal or external causes. The externally generated exceptions are the interrupts and the bus error and reset requests. The interrupts are requests from peripheral devices for processor action while the bus error and reset inputs are used for access control and processor restart. The internally generated exceptions come from instructions, or from address errors or tracing. The trap (TRAP), trap on overflow (TRAPV), check register against bounds (CHK) and divide (DIV) instructions all can generate exceptions as part of their instruction execution. In addition, illegal instructions, word fetches from odd addresses and privilege violations cause exceptions. Tracing behaves like a very high priority, internally generated interrupt after each instruction execution.

TABLE 18 — EXCEPTION VECTOR ASSIGNMENT

Vector Number(s)	Address			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, reserved)
13*	52	034	SD	(Unassigned, reserved)
14*	56	038	SD	(Unassigned, reserved)
15*	60	03C	SD	(Unassigned, reserved)
16-23*	64	04C	SD	(Unassigned, reserved)
	95	05F		—
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors
	191	0BF		—
48-63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		—
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		—

*Vector numbers 12 through 23 and 48 through 63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.

EXCEPTION PROCESSING SEQUENCE. Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the S-bit is asserted, putting the processor into the supervisor privilege state. Also, the T-bit is negated which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by a processor fetch, classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

The third step is to save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer. The program counter value stacked usually points to the next unexecuted instruction, however for bus error and address error, the value stacked for the program counter is unpredictable, and may be incremented from the address of the instruction which caused the error. Additional information defining the current context is stacked for the bus error and address error exceptions.

The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The processor then resumes instruction execution The instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution is started.

MULTIPLE EXCEPTIONS. These paragraphs describe the processing which occurs when multiple exceptions arise simultaneously. Exceptions can be grouped according to their occurrence and priority. The Group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to be aborted, and the exception processing to commence at the next minor cycle of the processor. The Group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instructions. These exceptions allow the current instruction to execute to completion, but preempt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed). The Group 2 exceptions occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and zero divide exceptions are in this group. For these exceptions, the normal execution of an instruction may lead to exception processing.

Group 0 exceptions have highest priority, while Group 2 exceptions have lowest priority. Within Group 0, reset has highest priority, followed by bus error and then address error. Within Group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at a time, there is no priority relation within Group 2.

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and the TRAP instruction processing is aborted. In another example, if an interrupt request occurs during the execution of an instruction while the T-bit is asserted, the trace exception has priority, and is processed first. Before instruction processing resumes, however, the interrupt exception is also processed, and instruction processing commences finally in the interrupt handler routine. A summary of exception grouping and priority is given in Table 19.

TABLE 19 — EXCEPTION GROUPING AND PRIORITY

Group	Exception	Processing
0	Reset Bus Error Address Error	Exception processing begins at the next minor cycle
1	Trace Interrupt Illegal Privilege	Exception processing begins before the next instruction
2	TRAP, TRAPV, CHK, Zero Divide	Exception processing is started by normal instruction execution

EXCEPTION PROCESSING DETAILED DISCUSSION

Exceptions have a number of sources, and each exception has processing which is peculiar to it. The following paragraphs detail the sources of exceptions, how each arises, and how each is processed.

RESET. The reset input provides the highest level exception. The processing of the reset signal is designed for system initiation, and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state, and the trace state is forced off. The processor interrupt priority mask is set at level seven. The vector number is internally generated to reference the reset exception vector at location 0 in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the supervisor stack pointer, neither the program counter nor the status register is saved. The address contained in the first two words of the reset exception vector is fetched as the initial supervisor stack pointer, and the address in the last two words of the reset exception vector is fetched as the initial program counter. Finally, instruction execution is started at the address in the program counter. The power-up/restart code should be pointed to by the initial program counter.

The RESET instruction does not cause loading of the reset vector, but does assert the reset line to reset external devices. This allows the software to reset the system to a known state and then continue processing with the next instruction.

INTERRUPTS. Seven levels of interrupt priorities are provided. Devices may be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the processor. Interrupt

priority levels are numbered from one to seven, level seven being the highest priority. The status register contains a three-bit mask which indicates the current processor priority, and interrupts are inhibited for all priority levels less than or equal to the current processor priority.

An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request lines; a zero indicates no interrupt request. Interrupt requests arriving at the processor do not force immediate exception processing, but are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current processor priority, execution continues with the next instruction and the interrupt exception processing is postponed. (The recognition of level seven is slightly different, as explained in a following paragraph).

If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. First a copy of the status register is saved, and the privilege state is set to supervisor, tracing is suppressed, and the processor priority level is set to the level of the interrupt being acknowledged The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. If external logic requests an automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number. If external logic indicates a bus error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing, saving the program counter and status register on the supervisor stack. The saved value of the program counter is the address of the instruction which would have been executed had the interrupt not been present. The content of the interrupt vector whose vector number was previously obtained is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine. A flow chart for the interrupt acknowledge sequence is given in Figure 28; a timing diagram is given in Figure 29.

FIGURE 28 — INTERRUPT ACKNOWLEDGE SEQUENCE FLOW CHART

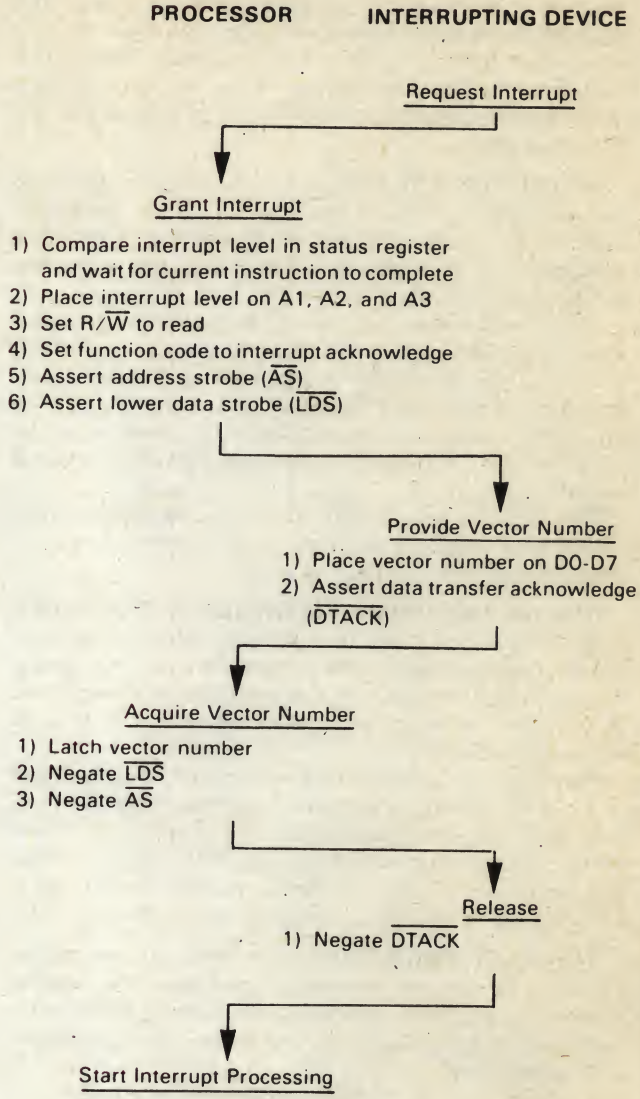
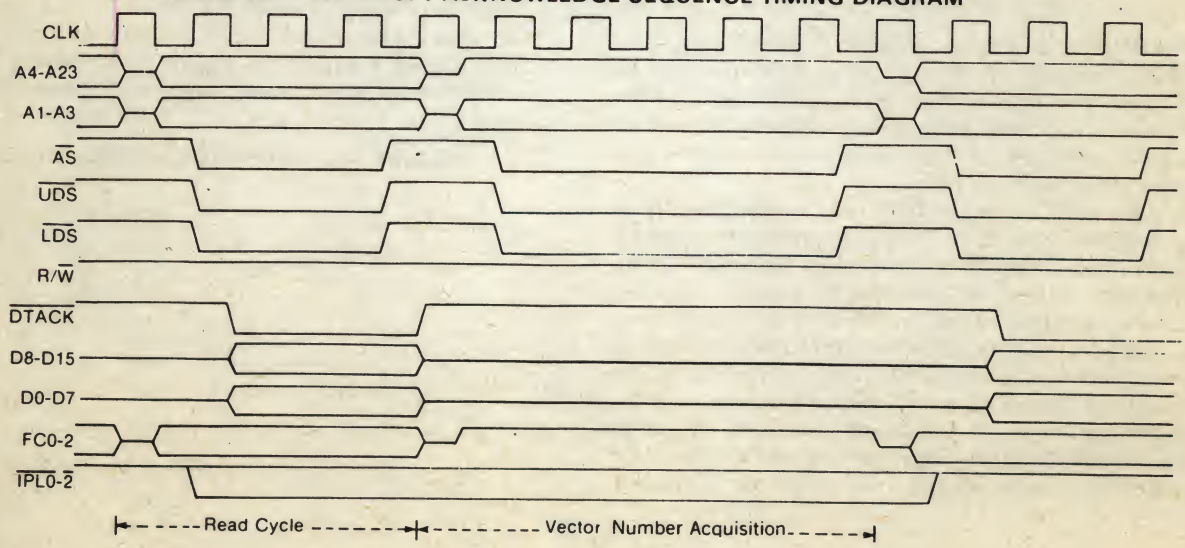


FIGURE 29 — INTERRUPT ACKNOWLEDGE SEQUENCE TIMING DIAGRAM



Priority level seven is a special case. Level seven interrupts cannot be inhibited by the interrupt priority mask, thus providing a "non-maskable interrupt" capability. An interrupt is generated each time the interrupt request level changes from some lower level to level seven. Note that a level seven interrupt may still be caused by the level comparison if the request level is a seven and the processor priority is set to a lower level by an instruction.

INSTRUCTION TRAPS. Traps are exceptions caused by instructions. They arise either from processor recognition of abnormal conditions during instruction execution, or from use of instructions whose normal behavior is trapping.

Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception, and is useful for implementing system calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a runtime error, which may be an arithmetic overflow or a subscript out of bounds.

The signed divide (DIVS) and unsigned divide (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

ILLEGAL AND UNIMPLEMENTED INSTRUCTIONS. Illegal instruction is the term used to refer to any of the word bit patterns which are not the bit pattern of the first word of a legal instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs.

Word patterns with bits 15 through 12 equaling 1010 or 1111 are distinguished as unimplemented instructions and separate exception vectors are given to these patterns to permit efficient emulation. This facility allows the operating system to detect program errors, or to emulate unimplemented instructions in software.

PRIVILEGE VIOLATIONS. In order to provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user state will cause an exception. The privileged instructions are:

STOP	AND (word) Immediate to SR
RESET	EOR (word) Immediate to SR
RTE	OR (word) Immediate to SR
MOVE to SR	MOVE USP

TRACING. To aid in program development, the MC68000 includes a facility to allow instruction by instruction tracing. In the trace state, after each instruction is executed an exception is forced, allowing a debugging program to monitor the execution of the program under test.

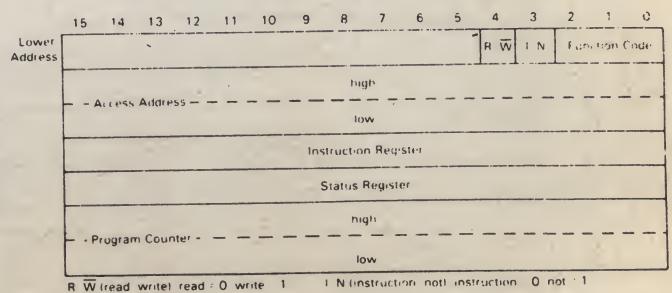
The trace facility uses the T-bit in the supervisor portion of the status register. If the T-bit is negated (off), tracing is disabled, and instruction execution proceeds from instruction to instruction as normal. If the T-bit is asserted (on) at the beginning of the execution of an instruction, a trace exception will be generated after the execution of that instruction is completed. If the instruction is not executed, either because an interrupt is taken, or the instruction is illegal or privileged, the trace exception does not occur. The trace exception also does not occur if the instruction is aborted by a reset, bus error, or address error exception. If the instruction is indeed executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. If, during the execution of the instruction, an exception is forced by that instruction, the forced exception is processed before the trace exception.

As an extreme illustration of the above rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt handler routine.

BUS ERROR. Bus error exceptions occur when external logic requests that a bus error be processed by an exception. The current bus cycle which the processor is making is aborted. Whether the processor was doing instruction or exception processing, that processing is terminated, and the processor immediately begins exception processing.

Exception processing for bus error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the bus error vector. Since the processor was not between instructions when the bus error exception request was made, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are of course saved. The value saved for the program counter is advanced by some amount, two to ten bytes beyond the address of the first word of the instruction which made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. Besides the usual information, the processor saves its internal copy of the first word of the instruction being processed, and the address which was being accessed by the aborted bus cycle. Specific information about the access is also saved: whether it was a read or a write, whether the processor was processing an instruction or not, and the classification displayed on the function code outputs when the bus error occurred. The processor is processing an instruction if it is in the normal state or processing a Group 2 exception; the processor is not processing an instruction if it is processing a Group 0 or a Group 1 exception. Figure 30 illustrates how this information is organized on the supervisor stack.

FIGURE 30 — SUPERVISOR STACK ORDER



Although this information is not sufficient in general to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in the vector. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, address error, or reset, the processor is halted, and all processing ceases. This simplifies the detection of catastrophic system failure, since the processor removes itself from the system rather than destroying all memory contents. Only the RESET pin can restart a halted processor.

ADDRESS ERROR. Address error exceptions occur when the processor attempts to access a word or a long word operand or an instruction at an odd address. The effect is much like an internally generated bus error, so that the bus cycle is aborted, and the processor ceases whatever processing it is currently doing and begins exception processing. After exception processing commences, the sequence is the same as that for bus error including the information that is stacked, except that the vector number refers to the address error vector instead. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted.

INTERFACE WITH M6800 PERIPHERALS

Motorola's extensive line of M6800 peripherals are directly compatible with the MC68000. Some of these devices that are particularly useful are:

MC6821 Peripheral Interface Adapter
MC6840 Programmable Timer Module
MC6843 Floppy Disk Controller
MC6845 CRT Controller
MC6849 Dual Density Floppy Disk Controller
MC6850 Asynchronous Communication Interface Adapter
MC6852 Synchronous Serial Data Adapter
MC6854 Advanced Data Link Controller
MC68488 General Purpose Interface Adapter

To interface the synchronous M6800 peripherals with the asynchronous MC68000, the processor modifies its bus cycle to meet the M6800 cycle requirements whenever an M6800 device address is detected. This is possible since both processors use memory mapped I/O. Figure 31 is a flow chart of the interface operation between the processor and M6800 devices

DATA TRANSFER OPERATION

Three signals on the processor provide the M6800 interface. They are: enable (E), valid memory address ($\overline{\text{VMA}}$), and valid peripheral address ($\overline{\text{VPA}}$). Figure 32 shows the timing required by M6800 peripherals, the timing specified for the M6800, and the corresponding timing for the MC68000. For further details on peripheral timing consult *The Complete Motorola Microcomputer Data Library*. Notice that the MC68000 $\overline{\text{VMA}}$ is active low, contrasted with the active high M6800 VMA. This allows the processor to put its buses in the high-impedance state on DMA requests without inadvertently selecting peripherals.

Enable corresponds to the E or $\phi 2$ signal in existing M6800 systems. It is the bus clock used by the M6800 peripherals to synchronize data transfer. Enable is a constant frequency clock that is one-tenth of the incoming MC68000 clock frequency. The timing of E allows 1 MHz peripherals to be used with an 8 MHz MC68000. Enable has a 60/40 duty cycle; that is, it is low for six input clocks and high for four input clocks. This duty cycle allows the processor to do two successive $\overline{\text{VPA}}$ accesses on successive E pulses.

FIGURE 31 — M6800 INTERFACING FLOW CHART

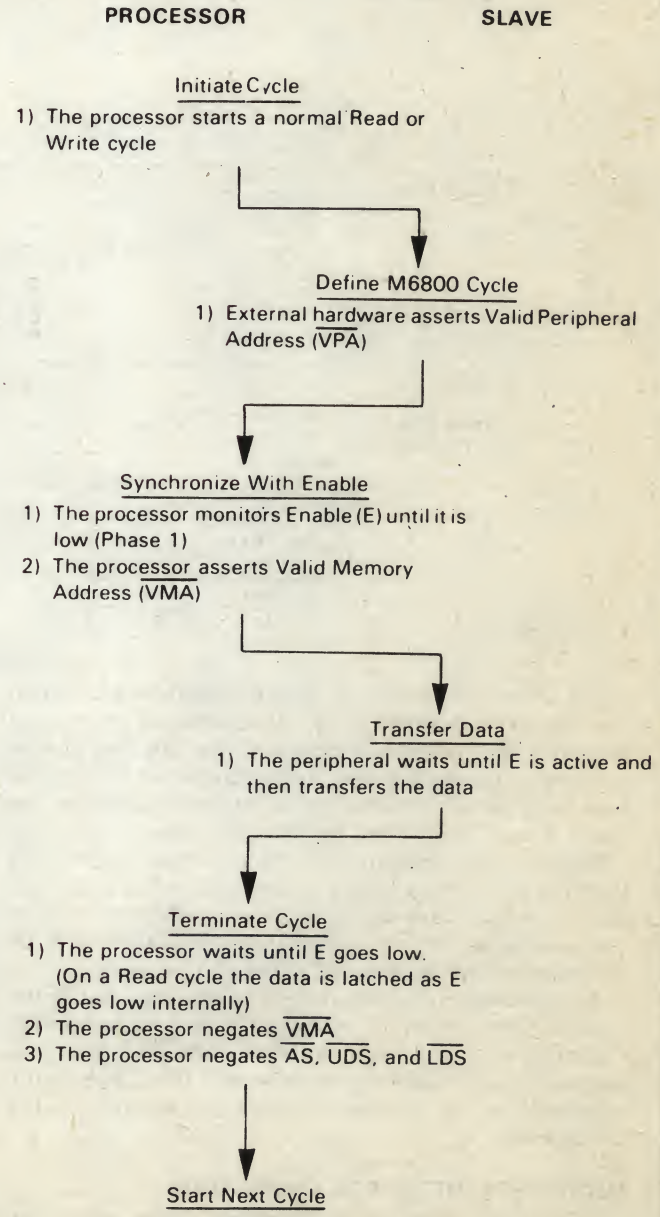
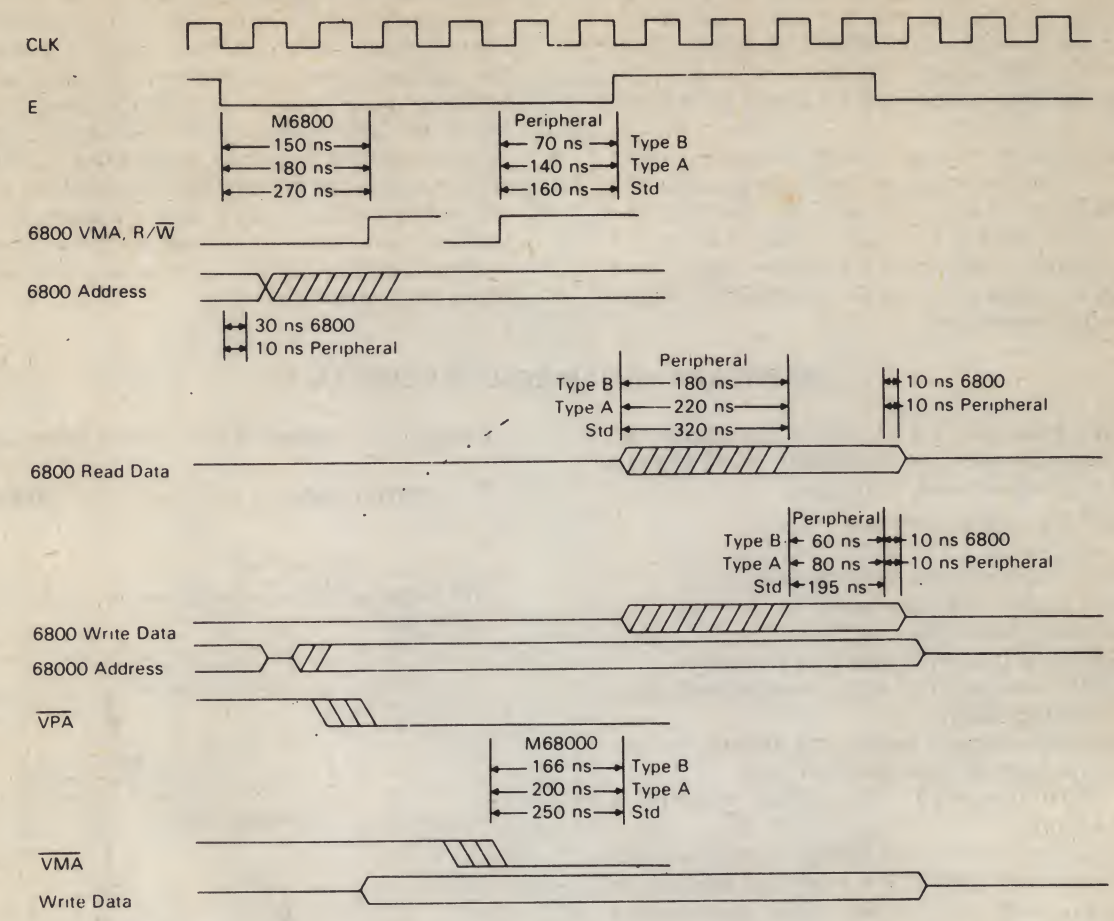


FIGURE 32 — MC68000 TO M6800 PERIPHERAL TIMING DIAGRAM



NOTE:
Times are expressed for different device clock frequencies.

CLK. FREQ.	F-TYPE
2.0 MHz	B
1.5 MHz	A
1.0 MHz	Std

Surplus Travel Time

F-TYPE	6800	68000
B	30 ns	96 ns
A	14 ns	60 ns
Std	70 ns	90 ns

The $\overline{VPA}$ input signals the processor that the address on the bus is the address of an M6800 device (or an area reserved for M6800 devices) and that the bus should conform to the $\phi 2$ transfer characteristics of the M6800 bus. Valid peripheral address is derived by decoding the address bus, conditioned by address strobe.

As a result of receiving $\overline{VPA}$, the processor issues $\overline{VMA}$ with the appropriate timing when E is low. Valid memory address is then used as part of the chip select equation of the peripheral. This ensures that the M6800 peripherals are selected and deselected at the correct time.

As is shown, the setup and hold times provided by the MC68000 are better than required. Surplus travel time is the delay that can be tolerated in the $\overline{VMA}$ line. It is defined as the difference between the $\overline{VMA}$ setup provided by the processor and that required by the peripheral.

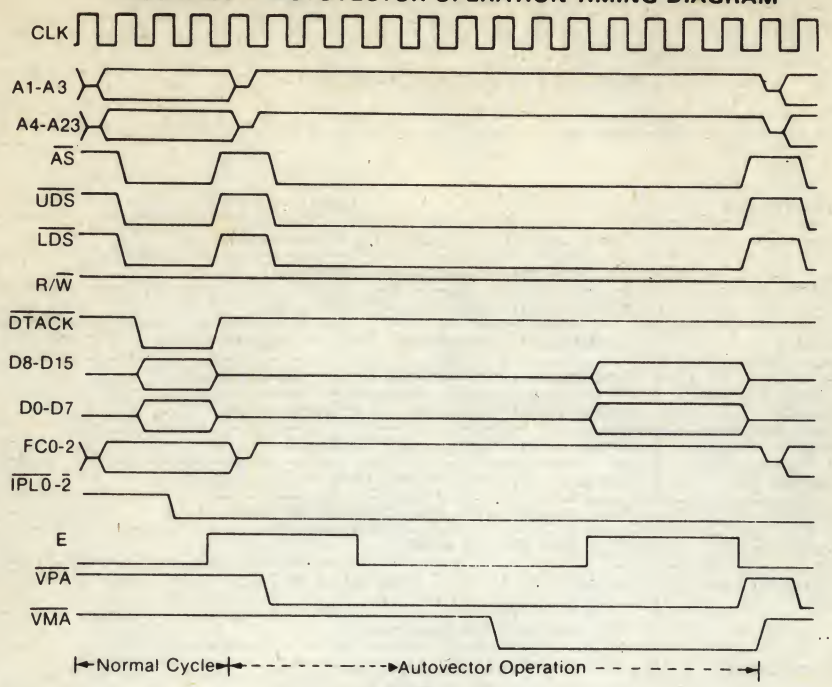
INTERRUPT INTERFACE OPERATION

During an interrupt acknowledge cycle while the processor is fetching the vector, if $\overline{VPA}$ is asserted, the MC68000 will assert $\overline{VMA}$ and complete a normal M6800 read cycle as shown in Figure 33. The processor will then use an internally generated vector that is a function of the interrupt being serviced. This process is known as autovectoring. The seven autovectors are vector numbers 25 through 31 (decimal).

This operates in the same fashion (but is not restricted to) the M6800 interrupt sequence. The basic difference is that there are six normal interrupt vectors and one NMI type vector. As with both the M6800 and the MC68000's normal vectored interrupt, the interrupt service routine can be located anywhere in the address space. This is due to the fact that while the vector numbers are fixed, the contents of the vector table entries are assigned by the user.

Since $\overline{VMA}$ is asserted during autovectoring, the M6800 peripheral address decoding should prevent unintended accesses.

FIGURE 33 — AUTOVECTOR OPERATION TIMING DIAGRAM



INSTRUCTION SET

The following paragraphs provide information about the addressing categories and instruction set of the MC68000.

ADDRESSING CATEGORIES

Effective address modes may be categorized by the ways in which they may be used. The following classifications will be used in the instruction definitions.

Data — If an effective address mode may be used to refer to data operands, it is considered a data addressing effective address mode.

Memory — If an effective address mode may be used to refer to memory operands, it is considered a memory addressing effective address mode.

Alterable — If an effective address mode may be used to refer to alterable (writable) operands, it is considered an alterable addressing effective address mode.

Control — If an effective address mode may be used to refer to memory operands without an associated size, it is considered a control addressing effective address mode.

Table 20 shows the various categories to which each of the effective address modes belongs. Table 21 is the instruction set summary.

The status register addressing mode is not permitted unless it is explicitly mentioned as a legal addressing mode.

These categories may be combined, so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses, and the latter refers to addressing modes which are both data and alterable.

TABLE 20 — EFFECTIVE ADDRESSING MODE CATEGORIES

Effective Address Modes	Mode	Register	Addressing Categories			
			Data	Memory	Control	Alterable
Dn	000	register number	X			X
An	001	register number				X
An@	010	register number	X	X	X	X
An@+	011	register number	X	X		X
An@-	100	register number	X	X		X
An@(d)	101	register number	X	X	X	X
An@(d, ix)	110	register number	X	X	X	X
xxx.W	111	000	X	X	X	X
xxx.L	111	001	X	X	X	X
PC@(d)	111	010	X	X	X	
PC@(d, ix)	111	011	X	X	X	
#xxx	111	100	X	X		

TABLE 21 — INSTRUCTION SET

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
ABCD	Add Decimal with Extend	(Destination)$_{10}$ + (Source)$_{10}$ ➔Destination	*	U	*	U	*
ADD	Add Binary	(Destination) + (Source) ➔Destination	*	*	*	*	*
ADDA	Add Address	(Destination) + (Source) ➔Destination	—	—	—	—	—
ADDI	Add Immediate	(Destination) + Immediate Data ➔Destination	*	*	*	*	*
ADDQ	Add Quick	(Destination) + Immediate Data ➔Destination	*	*	*	*	*
ADDX	Add Extended	(Destination) + (Source) + X ➔Destination	*	*	*	*	*
AND	AND Logical	(Destination) ∧ (Source) ➔Destination	—	*	*	0	0
ANDI	AND Immediate	(Destination) ∧ Immediate Data ➔Destination	—	*	*	0	0
ASL, ASR	Arithmetic Shift	(Destination) Shifted by <count> ➔Destination	*	*	*	*	*
Bcc	Branch Conditionally	If cc then PC + d ➔PC	—	—	—	—	—
BCHG	Test a Bit and Change	~(<bit number>) OF Destination ➔Z ~(<bit number>) OF Destination ➔ <bit number> OF Destination	—	—	*	—	—
BCLR	Test a Bit and Clear	~(<bit number>) OF Destination ➔Z 0 ➔<bit number > ➔OF Destination	—	—	*	—	—
BRA	Branch Always	PC + d ➔PC	—	—	—	—	—
BSET	Test a Bit and Set	~(<bit number>) OF Destination ➔Z 1 ➔<bit number> OF Destination	—	—	*	—	—
BSR	Branch to Subroutine	PC ➔SP@-; PC + d ➔PC	—	—	—	—	—
BTST	Test a Bit	~(<bit number>) OF Destination ➔Z	—	—	*	—	—
CHK	Check Register against Bounds	If Dn <0 or Dn> (<ea>) then TRAP	—	*	U	U	U
CLR	Clear an Operand	0 ➔Destination	—	0	1	0	0
CMP	Compare	(Destination) - (Source)	—	*	*	*	*
CMPA	Compare Address	(Destination) - (Source)	—	*	*	*	*
CMPI	Compare Immediate	(Destination) - Immediate Data	—	*	*	*	*
CMPM	Compare Memory	(Destination) - (Source)	—	*	*	*	*
DBcc	Test Condition, Decrement and Branch	If ~cc then Dn -1 ➔Dn; if Dn ≠ -1 then PC + d ➔PC	—	—	—	—	—
DIVS	Signed Divide	(Destination)/(Source) ➔Destination	—	*	*	*	0
DIVU	Unsigned Divide	(Destination)/(Source) ➔Destination	—	*	*	*	0
EOR	Exclusive OR Logical	(Destination)⊕(Source) ➔Destination	—	*	*	0	0
EORI	Exclusive OR Immediate	(Destination)⊕Immediate Data ➔Destination	—	*	*	0	0
EXG	Exchange Register	Rx ◄►Ry	—	—	—	—	—
EXT	Sign Extend	(Destination) Sign-extended ➔Destination	—	*	*	0	0
JMP	Jump	Destination ➔PC	—	—	—	—	—
JSR	Jump to Subroutine	PC ➔SP@-; Destination ➔PC	—	—	—	—	—
LEA	Load Effective Address	Destination ➔An	—	—	—	—	—
LINK	Link and Allocate	An ➔SP@-; SP ➔An; SP + d ➔SP	—	—	—	—	—
LSL, LSR	Logical Shift	(Destination) Shifted by <count> ➔Destination	*	*	*	0	*
MOVE	Move Data from Source to Destination	(Source) ➔Destination	—	*	*	0	0
MOVE to CCR	Move to Condition Code	(Source) ➔CCR	*	*	*	*	*
MOVE to SR	Move to the Status Register	(Source) ➔SR	*	*	*	*	*

* affected 0 cleared U undefined
— unaffected 1 set

TABLE 21 — INSTRUCTION SET (CONT.)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
MOVE from SR	Move from the Status Register	SR ➔Destination	—	—	—	—	—
MOVE USP	Move User Stack Pointer	USP ➔An; An ➔USP	—	—	—	—	—
MOVEA	Move Address	(Source) ➔Destination	—	—	—	—	—
MOVEM	Move Multiple Registers	Registers ➔Destination (Source) ➔Registers	—	—	—	—	—
MOVEP	Move Peripheral Data	(Source) ➔Destination	—	—	—	—	—
MOVEQ	Move Quick	Immediate Data ➔Destination	—	*	*	0	0
MULS	Signed Multiply	(Destination) * (Source) ➔Destination	—	*	*	0	0
MULU	Unsigned Multiply	(Destination) * (Source) ➔Destination	—	*	*	0	0
NBCD	Negate Decimal with Extend	0 - (Destination)$_{10}$ - X ➔Destination	*	U	*	U	*
NEG	Negate	0 - (Destination) ➔Destination	*	*	*	*	*
NEGX	Negate with Extend	0 - (Destination) - X ➔Destination	*	*	*	*	*
NOP	No Operation	—	—	—	—	—	—
NOT	Logical Complement	~(Destination) ➔Destination	—	*	*	0	0
OR	Inclusive OR Logical	(Destination) v (Source) ➔Destination	—	*	*	0	0
ORI	Inclusive OR Immediate	(Destination) v Immediate Data ➔Destination	—	*	*	0	0
PEA	Push Effective Address	Destination ➔SP@-	—	—	—	—	—
RESET	Reset External Devices	—	—	—	—	—	—
ROL, ROR	Rotate (Without Extend)	(Destination) Rotated by <count> ➔Destination	—	*	*	0	*
ROXL, ROXR	Rotate with Extend	(Destination) Rotated by <count> ➔Destination	*	*	*	0	*
RTE	Return from Exception	SP@+ ➔SR; SP@+ ➔PC	*	*	*	*	*
RTR	Return and Restore Condition Codes	SP@+ ➔CC; SP@+ ➔PC	*	*	*	*	*
RTS	Return from Subroutine	SP@+ ➔PC	—	—	—	—	—
SBCD	Subtract Decimal with Extend	(Destination) $_{10}$ - (Source)$_{10}$ - X ➔Destination	*	U	*	U	*
Scc	Set According to Condition	If cc then 1's ➔Destination else 0's ➔Destination	—	—	—	—	—
STOP	Load Status Register and Stop	Immediate Data ➔SR; STOP	*	*	*	*	*
SUB	Subtract Binary	(Destination) - (Source) ➔Destination	*	*	*	*	*
SUBA	Subtract Address	(Destination) - (Source) ➔Destination	—	—	—	—	—
SUBI	Subtract Immediate	(Destination) - Immediate Data ➔Destination	*	*	*	*	*
SUBQ	Subtract Quick	(Destination) - Immediate Data ➔Destination	*	*	*	*	*
SUBX	Subtract with Extend	(Destination) - (Source) - X ➔Destination	*	*	*	*	*
SWAP	Swap Register Halves	Register [31:16] ⇄Register [15:0]	—	*	*	0	0
TAS	Test and Set an Operand	(Destination) Tested ➔CC; 1 ➔[7] OF Destination	—	*	*	0	0
TRAP	Trap	PC ➔SSP@-; SR ➔SSP@-; (Vector) ➔PC	—	—	—	—	—
TRAPV	Trap on Overflow	If V then TRAP	—	—	—	—	—
TST	Test an Operand	(Destination) Tested ➔CC	—	*	*	0	0
UNLK	Unlink	An ➔SP; SP@+ ➔An	—	—	—	—	—

[ ] = bit number

INSTRUCTION EXECUTION TIMES

The following paragraphs contain listings of the instruction execution times in terms of the number of internal cycles required for an instruction to execute. One internal cycle is equal to two cycles of the processor clock input. It is also assumed that the memory cycle time is no greater than four cycles of the processor clock input to prevent the introduction of wait states in the bus cycle.

These timings are for the entire instruction, including operand fetches, operand writes, and instruction reads.

EFFECTIVE ADDRESS OPERAND CALCULATION TIMING

Table 22 gives the overhead time required for processing an effective address operand. The time includes fetching of any extension words, the address computation, and the fetching of the memory operand.

MOVE INSTRUCTION TIMING

Tables 23 and 24 give the timing for the move instruction. These are complete instruction timings, including operand reads and writes, and instruction reads.

TABLE 22 — EFFECTIVE ADDRESS OPERAND CALCULATION TIMING

Addressing Mode		Time	
		Byte, Word	Long
	Register		
Dn	Data Register Direct	0	0
An	Address Register Direct	0	0
	Memory		
An@	Address Register Indirect	2	4
An@+	Address Register Indirect with Postincrement	2	4
An@-	Address Register Indirect with Predecrement	3	5
An@(d)	Address Register Indirect with Displacement	4	6
An@(d, ix)	Address Register Indirect with Index	5	7
xxx.W	Absolute Short	4	6
xxx.L	Absolute Long	6	8
PC@(d)	Program Counter with Displacement	4	6
PC@(d, ix)	Program Counter with Index	5	7
#xxx	Immediate	2	4

TABLE 23 — MOVE BYTE AND WORD INSTRUCTION TIMING

Source	Destination								
	Dn	An	An@	An@+	An@-	An@(d)	An@(d, ix)	xxx.W	xxx.L
Dn	2	2	4	4	4	6	7	6	8
An	2	2	4	4	4	6	7	6	8
An@	4	4	6	6	6	8	9	8	10
An@+	4	4	6	6	6	8	9	8	10
An@-	5	5	7	7	7	9	10	9	11
An@(d)	6	6	8	8	8	10	11	10	12
An@(d, ix)	7	7	9	9	9	11	12	11	13
xxx.W	6	6	8	8	8	10	11	10	12
xxx.L	8	8	10	10	10	12	13	12	14
PC@(d)	6	6	8	8	8	10	11	10	12
PC@(d, ix)	7	7	9	9	9	11	12	11	13
#xxx	4	4	6	6	6	8	9	8	10

Note:
The move byte and word instructions each require one memory access for operand read and operand write.

TABLE 24 — MOVE LONG INSTRUCTION TIMING

Source	Destination								
	Dn	An	An@	An@+	An@-	An@(d)	An@(d, ix)	xxx.w	xxx.L
Dn	2	2	6	6	6	8	9	8	10
An	2	2	6	6	6	8	9	8	10
An@	6	6	10	10	10	12	13	12	14
An@+	6	6	10	10	10	12	13	12	14
An@-	7	7	11	11	11	13	14	13	15
An@(d)	8	8	12	12	12	14	15	14	16
An@(d, ix)	9	9	13	13	13	15	16	15	17
xxx.W	8	8	12	12	12	14	15	14	16
xxx.L	10	10	14	14	14	16	17	16	18
PC@(d)	8	8	12	12	12	14	15	14	16
PC@(d, ix)	9	9	13	13	13	15	16	15	17
#xxx	6	6	10	10	10	12	13	12	14

Note:
The move long instruction requires two memory accesses each for operand read and operand write.

STANDARD INSTRUCTION TIMING

The times shown in Table 25 include the time to perform the operation, store the results, and read the next instruction. Address calculation and effective address fetch time must be added where indicated. In Table 25, the headings have the following meanings: A = address register operand, D = data register operand, E = an operand specified by an effective address, and M = memory operand.

TABLE 25 — STANDARD INSTRUCTION TIMING

Instruction	Size	A op E →A	D op E →D	M op D →M
ADD	Byte, Word	4+	2+	4+
	Long	3#+	3#+	6+
AND	Byte, Word	—	2+	4+
	Long	—	3#+	6+
CMP	Byte, Word	3+	2+	—
	Long	3+	3+	—
DIVS	—	—	79+*	—
DIVU	—	—	70+*	—
EOR	Byte, Word	—	2	4+
	Long	—	3#	5+
MULS	—	—	35+*	—
MULU	—	—	35+*	—
OR	Byte, Word	—	2+	4+
	Long	—	3#+	6+
SUB	Byte, Word	4+	2+	4+
	Long	3#+	3#+	6+

\+ add effective address calculation time
\# 4 if the effective address is register direct
* indicates maximum value

IMMEDIATE INSTRUCTION TIMING

The times shown in Table 26 include the time to fetch immediate operands, perform the operations, store the results, and read the next operation. In Table 26, the headings have the following meanings: D = data register operand, I = immediate operand, M = memory operand, SR = status register.

TABLE 26 — IMMEDIATE INSTRUCTION TIMING

Instruction	Size	D op I →D	M op I →M	SR op I →SR
ADDI	Byte, Word	4	6+	—
	Long	8	10+	—
ADDQ	Byte, Word	2	4+	—
	Long	4	6+	—
ANDI	Byte, Word	4	6+	10
	Long	8	10+	—
CMPI	Byte, Word	4	4+	—
	Long	7	6+	—
EORI	Byte, Word	4	6+	10
	Long	8	10+	—
MOVEQ	Long	2	—	—
ORI	Byte, Word	4	6+	10
	Long	8	10+	—
SUBI	Byte, Word	4	6+	—
	Long	8	10+	—
SUBQ	Byte, Word	2	4+	—
	Long	4	6+	—

\+ add effective address calculation time

SINGLE OPERAND INSTRUCTION TIMING

Table 27 gives the timing for single operand instructions.

TABLE 27 — SINGLE OPERAND INSTRUCTION TIMING

Instruction	Size	Register	Memory
CLR	Byte, Word Long	2 3	4+ 6+
NBCD	Byte	3	4+
NEG	Byte, Word Long	2 3	4+ 6+
NEGX	Byte, Word Long	2 3	4+ 6+
NOT	Byte, Word Long	2 3	4+ 6+
Scc	Byte, False Byte, True	2 3	4+ 4+
TAS	Byte	2	5+
TST	Byte, Word Long	2 2	2+ 2+

+ add effective address calculation time

SHIFT/ROTATE INSTRUCTION TIMING

Table 28 gives the timing for shift and rotate instructions.

TABLE 28 — SHIFT AND ROTATE INSTRUCTION TIMING

Instruction	Size	Register*	Memory
ASR, ASL	Byte, Word Long	3 + n 4 + n	4+ —
LSR, LSL	Byte, Word Long	3 + n 4 + n	4+ —
ROR, ROL	Byte, Word Long	3 + n 4 + n	4+ —
ROXR, ROXL	Byte, Word Long	3 + n 4 + n	4+ —

+ add the effective address calculation time
*n = the shift count

BIT MANIPULATION INSTRUCTION TIMING

Table 29 gives the timing for the bit manipulation instructions.

TABLE 29 — BIT MANIPULATION INSTRUCTION TIMING

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Word Long	— 4*	4+ —	— 6*	6+ —
BCLR	Word Long	— 5*	4+ —	— 7*	6+ —
BSET	Word Long	— 4*	4+ —	— 6*	6+ —
BTST	Word Long	— 3	2+ —	— 5	4+ —

+ add effective address calculation time
* indicates maximum value

CONDITIONAL INSTRUCTION TIMING

Table 30 gives the timing for the conditional instructions.

TABLE 30 — CONDITIONAL INSTRUCTION TIMING

Instruction	Displacement	Successful (TRAP)	Unsuccessful (No TRAP)
Bcc	byte word	5 5	4 6
BRA	byte word	5 5	— —
BSR	byte word	9 9	— —
DBcc	—	5	7
CHK	—	19*+	4+
TRAP	—	16	—
TRAPV	—	16	2

+ add effective address calculation time
* indicates maximum value

JMP, JSR, LEA, PEA, MOVEM INSTRUCTION TIMING

Table 31 gives the timing for the jump, jump to subroutine, load effective address, push effective address, and move multiple registers.

TABLE 31 — JMP, JSR, LEA, PEA, MOVEM INSTRUCTION TIMING

Instruction	Size	An@	An@+	An@-	An@(d)	An@(d, ix)	xxx.W	xxx.L	PC@(d)	PC@(d, ix)
JMP	—	4	—	—	5	7	5	6	5	7
JSR	—	8	—	—	9	11	9	10	9	11
LEA	—	2	—	—	4	6	4	6	4	6
MOVEM	Word	6 + 2n	6 + 2n	—	8 + 2n	9 + 2n	8 + 2n	10 + 2n	8 + 2n	9 + 2n
M →R	Long	6 + 4n	6 + 4n	—	8 + 4n	9 + 4n	8 + 4n	10 + 4n	8 + 4n	9 + 4n
MOVEM	Word	4 + 2n	—	4 + 2n	6 + 2n	7 + 2n	6 + 2n	8 + 2n	—	—
R →M	Long	4 + 4n	—	4 + 4n	6 + 4n	7 + 4n	6 + 4n	8 + 4n	—	—
PEA		6	—	—	8	9	8	10	8	9

n = the number of registers to move

EXCEPTION PROCESSING TIMING

Table 32 gives the timing for exception processing. These times include all automatic stacking, the vector fetch, and the fetch of the first instruction of the handler routine.

TABLE 32 — EXCEPTION PROCESSING TIME

Exception	Time
Address Error	24
Bus Error	24
Interrupt	21*
Illegal Instruction	16
Privileged Instruction	16
Trace	16

* Interrupt acknowledge is assumed to take two cycles.

MULTI-PRECISION INSTRUCTIONS TIMING

Table 33 gives the timings for the multi-precision instructions. These times include the time to fetch both operands, perform the operations, store the results, and read the next instructions.

TABLE 33 — MULTI-PRECISION INSTRUCTIONS TIMING

Instruction	Size	D op D →D	M op M →M
ADDX	Byte, Word	2	9
	Long	4	15
CMPM	Byte, Word	—	6
	Long	—	10
SUBX	Byte, Word	2	9
	Long	4	15
ABCD	Byte	3	9
SBCD	Byte	3	9

MISCELLANEOUS INSTRUCTIONS TIMING

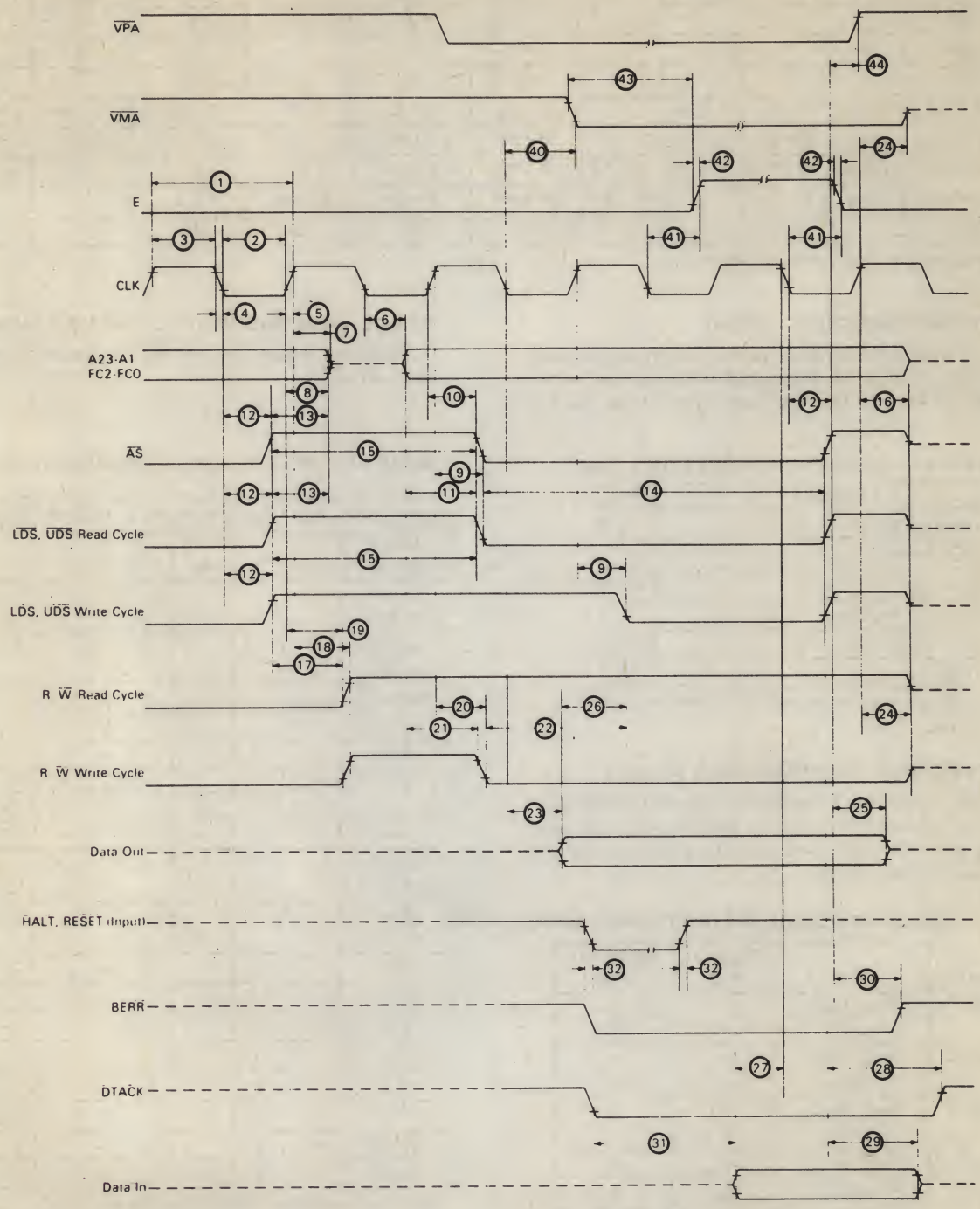
Table 34 gives the timing for some miscellaneous instructions.

TABLE 34 — MISCELLANEOUS INSTRUCTIONS TIMING

Instruction	Size	Reg.	Mem.	Reg. →Mem.	Mem. →Reg.
MOVE from SR	—	3	4+	—	—
MOVE to CCR	—	6	6+	—	—
MOVE to SR	—	6	6+	—	—
MOVEP	Word	—	—	8	8
	Long	—	—	12	12
EXG	—	3	—	—	—
EXT	Word	2	—	—	—
	Long	2	—	—	—
LINK	—	8	—	—	—
MOVE from USP	—	2	—	—	—
MOVE to USP	—	2	—	—	—
NOP	—	2	—	—	—
RESET	—	66	—	—	—
RTE	—	10	—	—	—
RTR	—	10	—	—	—
RTS	—	8	—	—	—
STOP	—	2	—	—	—
SWAP	—	2	—	—	—
UNLK	—	6	—	—	—

\+ add effective address calculation time

AC ELECTRICAL WAVEFORMS



NOTES:

1. This timing diagram should only be referenced in regard to the edge-to-edge measurement of timing specifications given in the AC Electrical Specifications. It is not intended as a functional description of the inputs and outputs. Refer to other functional descriptions and their related waveform diagrams for device operation. Bus operation is assumed to have a four cycle READ and a four cycle WRITE in this diagram

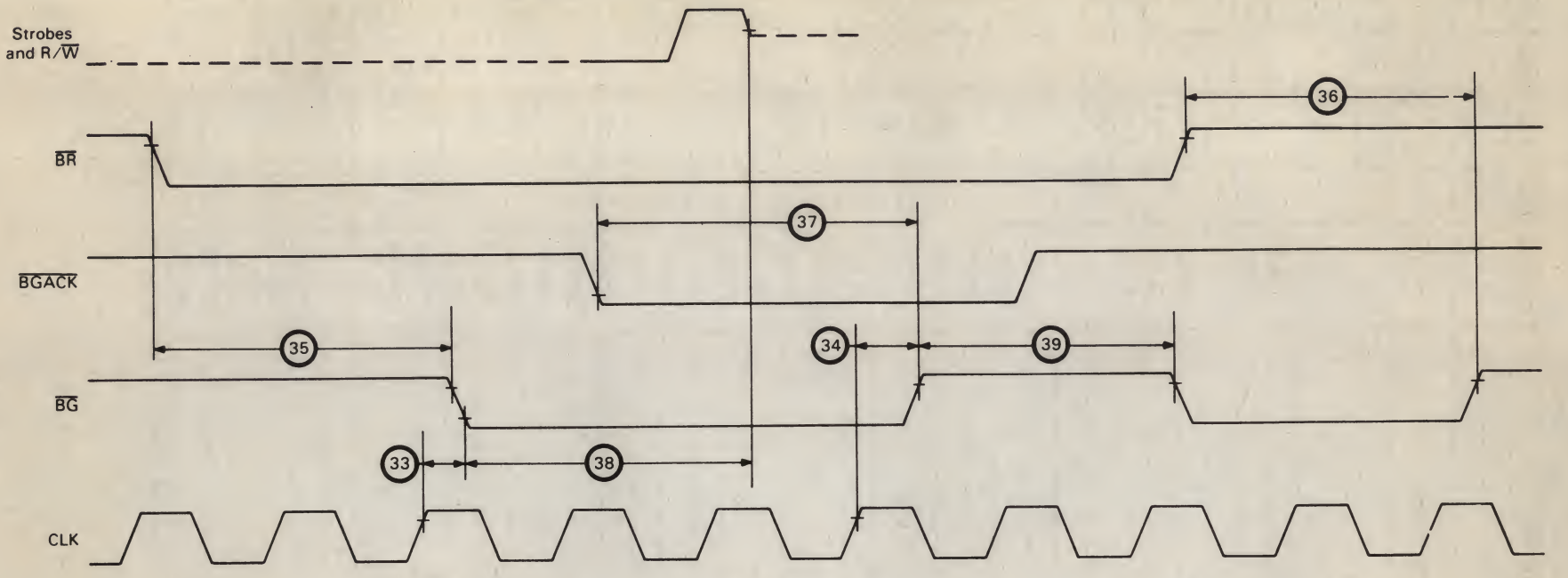
2. Waveform measurements are specified:
for all outputs as: V_{OH} = 2.4 volts
V_{OL} = 0.4 volts
for all inputs as: V_{IH} = 2.0 volts
V_{IL} = 0.8 volts
Exception is made when the measurement is made to the high impedance level, then 0.5 volts is subtracted from V_{OH} and 0.5 volts is added to V_{OL}.

AC ELECTRICAL SPECIFICATIONS (V_{DD} = 5.0 Vdc ±5%; V_{SS} = 0 Vdc; T_A = 25°C)

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Clock Period	t_{CYC}	—	125	—	ns
2	Clock Width Low	t_{CL}	—	55	—	ns
3	Clock Width High	t_{CH}	—	55	—	ns
4	Clock Fall Time	t_{Cf}	—	—	10	ns
5	Clock Rise Time	t_{Cr}	—	—	10	ns
6	Clock Low to Address/FC Valid	t_{CLAV}	—	70	—	ns
7	Clock High to Address/FC High Impedance (max)	t_{CHAZx}	—	70	—	ns
8	Clock High to Address/FC Invalid (min)	t_{CHAZn}	—	20	—	ns
9	Clock High to $\overline{AS}$, $\overline{DS}$ Low (max)	t_{CHSLx}	—	60	—	ns
10	Clock High to $\overline{AS}$, $\overline{DS}$ Low (min)	t_{CHSLn}	—	20	—	ns
11	Address/FC Valid to $\overline{AS}$, $\overline{DS}$ (read) Low	t_{AVSL}	—	30*	—	ns
12	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}	—	40	—	ns
13	$\overline{AS}$, $\overline{DS}$ High to Address/FC Invalid	t_{SHAZ}	—	40*	—	ns
14	$\overline{AS}$, $\overline{DS}$ Width Low (read)	t_{SL}	—	150	—	ns
15	$\overline{AS}$, $\overline{DS}$ Width High	t_{SH}	—	150	—	ns
16	Clock High to $\overline{AS}$, $\overline{DS}$ High Impedance	t_{CHSZ}	—	60	—	ns
17	$\overline{DS}$ High to R/$\overline{W}$ High	t_{SHRH}	—	60*	—	ns
18	Clock High to R/$\overline{W}$ High (max)	t_{CHRHx}	—	60	—	ns
19	Clock High to R/$\overline{W}$ High (min)	t_{CHRHn}	—	20	—	ns
20	Clock High to R/$\overline{W}$ Low	t_{CHRL}	—	60	—	ns
21	Address/FC Valid to R/$\overline{W}$ Low	t_{AVRL}	—	50*	—	ns
22	R/$\overline{W}$ Low to $\overline{DS}$ Low (write)	t_{RLSL}	—	80*	—	ns
23	Clock Low to Data Out Valid	t_{CLDO}	—	50	—	ns
24	Clock High to R/$\overline{W}$, $\overline{VMA}$ High Impedance	t_{CHRZ}	—	60	—	ns
25	$\overline{DS}$ High to Data Out Invalid	t_{SHDO}	—	30*	—	ns
26	Data Out Valid to $\overline{DS}$ Low (write)	t_{DOSL}	—	30*	—	ns
27	Data In to Clock Low (set up time)	t_{DICL}	—	30	—	ns
28	$\overline{DS}$ High to $\overline{DTACK}$ High	t_{SHDAH}	0	*	120	ns
29	$\overline{DS}$ High to Data In (hold time)	t_{SHDI}	0	—	—	ns
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	t_{SHBEH}	0	—	—	ns
31	$\overline{DTACK}$ Low to Data In (setup time)	t_{DALDI}	—	90*	—	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	t_{RHrf}	—	—	200	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	—	60	—	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	—	60	—	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	—	3.0	clk. per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	—	3.0	clk. per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	—	2.0	clk. per.
38	$\overline{BG}$ Low to Bus High Impedance (with $\overline{AS}$ high)	t_{GLZ}	0	—	—	clk. per.
39	$\overline{BG}$ Width High	t_{GH}	1.5	—	—	clk. per.
40	Clock Low to $\overline{VMA}$ Low	t_{CLVML}	—	60	—	ns
41	Clock Low to E Transition	t_{CLE}	—	55	—	ns
42	E Output Rise and Fall Time	t_{Erf}	—	—	25	ns
43	$\overline{VMA}$ Low to E High	t_{VMLEH}	2.0	—	3.0	clk. per.
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	t_{SHVPH}	0	—	—	ns

*Actual value dependent upon actual clock period. These figures are based on 8 MHz operation.

AC ELECTRICAL WAVEFORMS BUS ARBRITRATION



AC ELECTRICAL SPECIFICATIONS (V_{DD} = 5.0 Vdc ±5%; V_{SS} = 0 Vdc; T_A = 25°C) BUS ARBITRATION

Number	Characteristic	Symbol	Min	Typ	Max	Unit
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	—	60	—	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	—	60	—	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.0	—	3.0	clk. per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.0	—	3.0	clk. per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.0	—	2.0	clk. per.
38	$\overline{BG}$ Low to Bus High Impedance (with $\overline{AS}$ high)	t_{GLZ}	0	—	1.5	clk. per.
39	$\overline{BG}$ Width High	t_{GH}	1.5	—	—	clk. per.

MACSbug COMMAND SUMMARY

COMMAND	DESCRIPTION
reg#	Print a register
reg# hexdata	Put a hex value in the register
reg# 'ASCII'	Put hex-equivalent characters in register
reg#:	Print the old value and request new value
class	Print all registers of a class (A or D)
class:	Sequence through-print old value request new
DM start end	Display memory, hex-ASCII memory dump
SM address data	Set memory with data
OPen address	Open memory for read/change
SYmbol NAME value	Define and print symbols
W#	Print the effective address of the window
##. len EA	Define window length and address mode
M# data	Memory in window, same syntax as register
Go	Start running from address in program counter
Go address	Start running from this address
Go TILL add	Set temporary breakpoint and start running
BReakpoint	Print all breakpoint addresses
BR add: count	Set a new breakpoint and optional count
BR —address	Clear a breakpoint
BR CLEAR	Clear all breakpoints
TD	Print the trace display
TD reg#. format	Put a register in the display
TD Clear	Take all registers out of the display
TD ALI	Set all registers to appear in the display
TD A. 1 D. 1 L. c	Set register blocks or line separator
T	Trace one instruction
T count	Trace the specified number of instructions
T TILL Address	Trace until this address
:*(CR)	Carriage return-trace one instruction
OFfset address	Define the global offset
CV decimal	Convert decimal number to hex
CV $hex	Convert hex to decimal
CV value,value	Calculate offset or displacement
REad ; =text	Expect to receive 'S' records
VErify ; =text	Check memory against 'S' records
PUnch start end	Print 'S' records (tape image)
FOrmat hex	Program/initalize an ACIA
NUll hex	Set character null pads
CR hex	Set carriage return null pads
TErminal baud	Set terminal null pads to default values
CAll address	JSR to user utility routine
P2	Enter transparent mode
*..data..	Transmit command to host

MANUDAX LEVERT UIT VOORRAAD

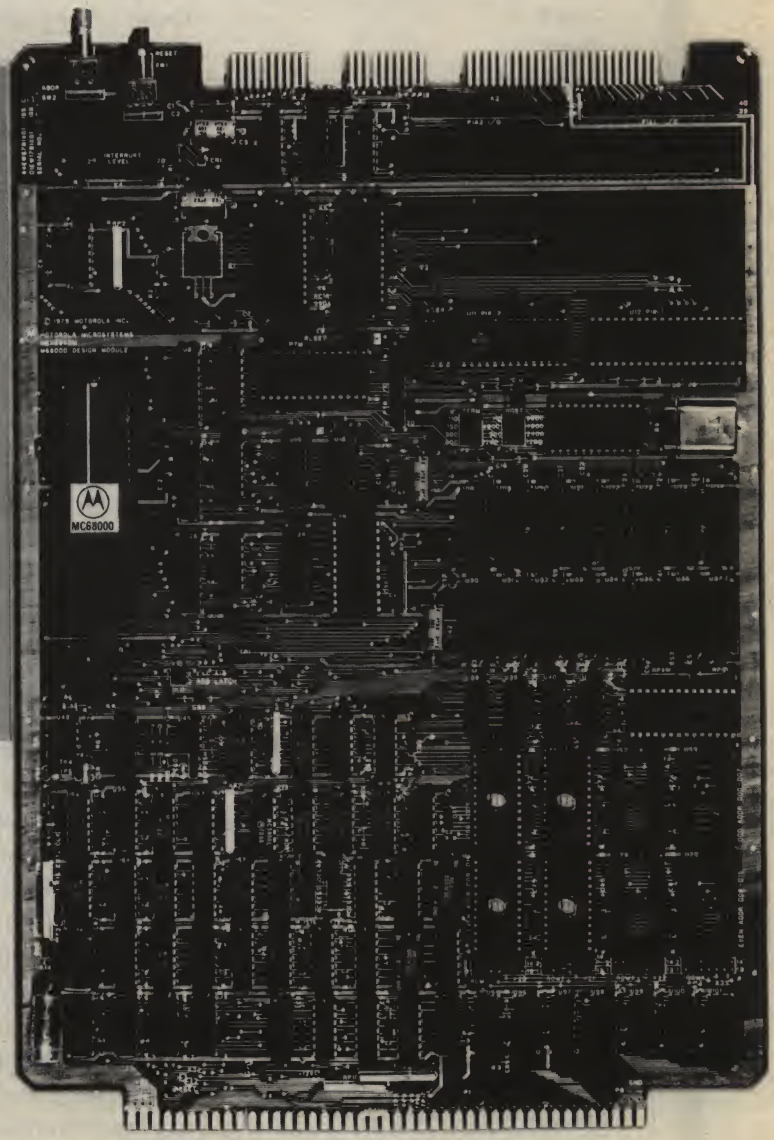
THE M68000 DESIGN MODULE

Motorola Introduces MC68000 Design Module

Phoenix, Arizona . . . Concurrent with the announcement of its new MC68000 microprocessor, Motorola is introducing a MC68000 Design Module — MEX68KDM.

An easy and convenient means for evaluating and designing with the MC68000, the design module permits chip evaluation, using either a Motorola development system, or an IBM370 or PDP11 host computer in conjunction with cross computer software. It is also suitable for OEM applications with the MC68000 by virtue of on-board 32K bytes of RAM, two 16-bit parallel I/O ports, three 16-bit programmable timers, two serial RS-232 ports and 8K bytes of debug ROM.

MEX68KDM includes MACSbug, one of the most powerful 16-bit microprocessor debug tools available. Once a memory file is resident in RAM, a user may begin his program bebug phase using the extensive MACSbug command structure provided in ROM . . .



. . . The Design Module is 14 × 9-3/4 inches making it compatible with Motorola's EXORciser and Micromodule Chassis. Power connections as well as the MC68000's buffered address and data bus appear at the EXORciser-compatible edge connector. However, the address and data bus may be electrically isolated from the edge connector, permitting the Design Module to obtain power from an EXORciser Chassis while maintaining functional independance for both the EXORciser and the 68000-based Design Module.

The MEX68KDM board contains 32K bytes of RAM. The on-board RAM block is configured as 16K 16-bit words matching the 68000 Microprocessor's data bus. It is made from 16K × 1 Dynamic RAMS (16 total) with refresh of the Dynamic RAM handled by on-board logic.

The Design Module also contains a 4K-word monitor ROM, called MACSbug, which is used to load and interact with user programs. This interaction consists of examining and initializing RAM memory locations, examining and initializing MC68000 registers and allowing the execution of user programs. MACSbug resides in 4 of the 8 ROM sockets on the MEX68KDM